

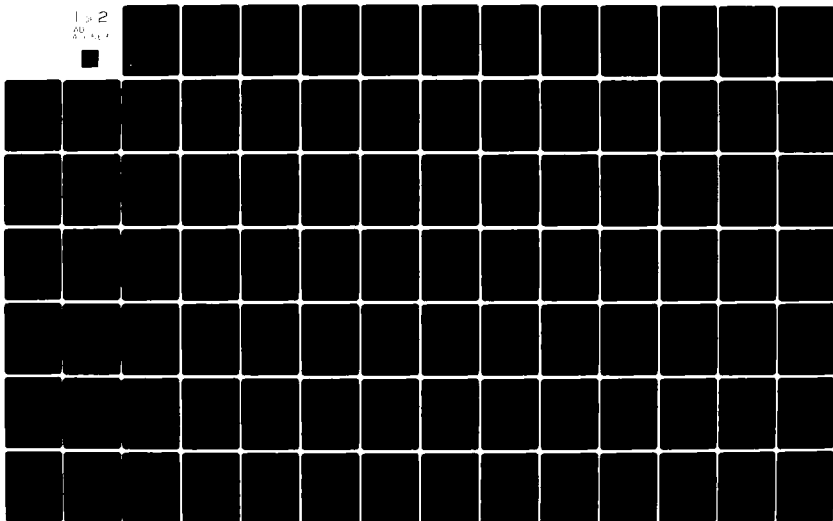
AD-A111 563

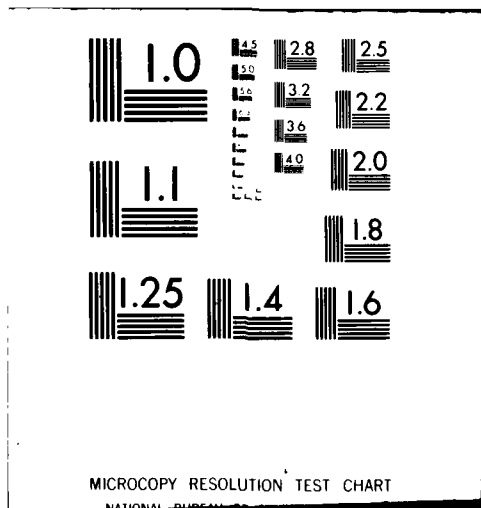
FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 17/2
(KSOS) KERNEL VERIFICATION RESULTS. KERNELIZED SECURE OPERATING--ETC(U)
DEC 80

UNCLASSIFIED WDL-TR9001

NL

1 of 2
AD
6 1 1 1 1





②

ADA 111563

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS) KERNEL VERIFICATION RESULTS.

Department of Defense Kernelized Secure Operating System

Contract MDA 903-77-C-0333
CDRL 0002BG

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

DTIC
ELECTE
MAR 03 1982
E

DTIC FILE COPY



Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

Approved for public release; distribution unlimited.

82 03 03 031

CONTENTS

1. Introduction	1
2. KSOS Verification Achievements	2
2.1 The MLS Formula Generator	2
2.2 Proof of the Kernel Specifications	2
2.2.1 Analysis of the Specification Proofs	7
2.2.2 Additional Lessons Learned from the Kernel Specification Proofs	7
2.3 Prototype Tools for Specification and Code Proofs	8
2.3.1 Summary of Tools and Manual Steps	12
2.3.2 Additional Notes	13
2.3.3 Sample MLS Tool Outputs and Their Interpretation	13
2.4 Code Proof for a Simplified Module	15
2.5 Manual Code/Specification Analysis	15
3. Non-Achievements	17
4. Appendix A - Kernel Formal Specifications	18
5. Appendix B - Sample Code Proof	19

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or Special
A	



KSOS Kernel Verification Results

1. Introduction

The original KSOS verification goals were the following:

1. The instantiation of the multilevel security model to SPECIAL;
2. The design and development of a computer tool (the MLS formula generator) whose input would be SPECIAL specifications, and whose output would be conjectures, the proof of which would imply that the specifications do not contain any violations of the multilevel security model;
3. The proofs that the specifications for the KSOS security perimeter (Kernel and NKSR) conform to the security model. In the event of violations, the proof process should pinpoint the violations so that they may be eliminated or bandwidth-limited;
4. The development of support tools so that illustrative code proofs could be carried out. The goal of these code proofs is to demonstrate the feasibility of performing full code proofs at a later date, and
5. The carrying out of illustrative code proofs.

KSOS has succeeded in instantiating the multilevel security model to SPECIAL and developing the MLS formula generator, in producing proofs of the kernel specifications, in producing prototype support tools that could be used in code proofs, in producing a code proof for a version of the SMXflow module, and in producing some mapping functions (manually) showing the correspondence of VFUNS to Modula structures for certain kernel modules. Due to a variety of organizational and technical factors indicated below, KSOS has been less than successful in producing specification proofs for the NKSR, in producing human-engineered, fully documented support tools for code proofs, and in producing code proofs other than for simplified modules.

KSOS Kernel Verification Results

2. KSOS Verification Achievements

This section provides details of the achievements mentioned above, and indicates their benefits.

2.1 The MLS Formula Generator

The instantiation of the MLS model to SPECIAL, and a description of the MLS formula generator are found in the report "A Technique for Proving Specifications are Multilevel Secure", SRI CSL-109, January 1980. Although this tool has limitations, as mentioned below, the concept upon which it is based represents an important breakthrough in verifying security properties of systems. Its main virtue is that the designer of a system need only supply the specifications of the system as input; in contrast to other verification systems (e.g., INA JO), it is not necessary to supply additional assertions. Several current limitations and areas for improvement are mentioned in the above-cited report: a variety of restrictions of SPECIAL are imposed on the designer; the semantics of SPECIAL is defined only within the code for the formula generator and in the definitions and axioms of the theorem prover; the human interface is clumsy and provides little help in analyzing the output; and it is largely ad-hoc in construction and behavior creating the possibility of failed proofs that should succeed and unsound proofs in the face of security flaws. In addition to this list, we have noted in our utilization of the tool that a large majority of formulas that are sent to the theorem prover are not much more complex than $x \leq x$, which might be filtered out by a more powerful simplifier.

2.2 Proof of the Kernel Specifications

Between November 1979 and February 1980 there was intensive activity subjecting the kernel specification to analysis using the MLS and Theorem Proving tools. The specifications for all 34 top level kernel calls, and their supporting specifications, were processed. Space limitations prevented the entire kernel from being processed in a single run, so the kernel specifications were broken into 5 modules. It is significant that the modularity of the kernel specifications permitted such a decomposition after the fact.

In the first run, 24 November 1979, a total of 1654 formulas were generated by MLS. 755 of these were sufficiently trivial so that the MLS tool was able to deduce their validity without passing them on to the Theorem Prover. Of the remaining 899 formulas, 586 were proved by the Theorem Prover, and 313 were unproved. In the final run, 5 February 1980, the figures were: 1598 formulas generated, 867 proved trivially by MLS, 416 proved by Theorem Prover, and 315 unproved. Detailed charts showing the statistics for each kernel call are presented below. These specifications are included as Appendix A.

In the remainder of this section, we will discuss the significance of these numbers, and the effect that the runs had on subsequent modifications to the specifications. There is no significant correlation between the number of unproved conjectures and the degree to which the specifications contain security violations. This is because a subtle and deep violation of the security model may generate a small number of conjectures, whereas a simple and easily

KSOS Kernel Verification Results

repairable violation may generate many unproved conjectures. Nonetheless, the totality of unproved conjectures is significant in that it maps onto the totality of violations of the model.

The first several runs pinpointed problems in the MLS tool itself and in the style of writing specifications. The MLS tool was unable to handle resource errors, renaming, and produced numerous duplicate formulas to be sent to the Theorem Prover. In terms of specification writing style, the tool had trouble with EFFECTS OF clauses, with ordering of exceptions, and (somewhat to our amazement) with treating logically equivalent Boolean expressions equivalently (e.g. AND and OR are treated nonsymmetrically, with the result that DeMorgan's laws do not apply as far as the MLS tool is concerned). Therefore phase 1 of our efforts dealt with rewriting the specifications to work around these problems, and with the correction of certain difficulties with the MLS tool.

The next phase of our involvement with the specifications dealt with adding knowledge that the Theorem Prover would need to prove some of the unproved conjectures. For example, various security properties hold for open files, because such properties were checked at the time the file was opened, and no security changes were made since then. Such information was added in the form of axioms (or unproved lemmas), which allowed some of the unproved conjectures to be proved. However, extreme caution is needed in adding axioms (i) to avoid adding inconsistencies (in which case every conjecture is provable, due to the interesting property of logical implication, that an inconsistency implies FALSE, and FALSE implies anything), and (ii) to avoid adding consistent but unwarranted axioms. It is very unlikely that an inconsistency would be added that would not be detected subsequently by human analysis. For several runs, however, there were some unwarranted axioms to the effect that an object can only access objects at the same security level (such a system is known as "stratified"). As a result, several conjectures were proven which should not have been.

Thus, the discussion has focused on experiences with the tools themselves, and with adding supplemental knowledge in the form of axioms. The analysis of the unproved conjectures is also significant. We categorized the reasons for failing to prove a conjecture into the following:

- ♦ resource error (representing a potential channel, based on resource utilization patterns, which may be bandwidth-limited by the introduction of random delays in the implementation, rather than eliminated);
- ♦ errors that can be fixed by redesigning the kernel;
- ♦ errors that are allowed to remain because there is no way for illicit information to leak beyond the security perimeter (including deliberate violations in the trusted software, needed to achieve desired functionality, as well as violations to hidden VFUNS);
- ♦ and valid formulas which the theorem prover could not prove.

We have analyzed every error detected by the tools, and taken appropriate action. However, the funding for continued utilization of the tools on live

KSOS Kernel Verification Results

specifications was depleted before the final version of the specifications was produced. Thus, potentially, there are violations in the final version of the specifications that should be fixed.

Next are presented detailed charts showing the statistics for each of the 34 kernel calls in terms of number of formulas generated (FOR), trivially proved (TRV), proved by Theorem Prover (THM), and unproved (UNP). Following these charts, the next several subsections deal with a more detailed analysis of the changes made to the specifications as a result of analyzing the output of the tools.

KSOS Kernel Verification Results

11/19/79 Status of KSOS Specifications

	FOR	TRV	THM	UNP
(KER1)				
K_FORK	158	39	18	101
K_GET_PROCESS_STATUS	2	0	2	0
K_INTERRUPT_RETURN	9	9	0	0
K_INVOKE	61	42	19	0
K_NAP	0	0	0	0
K_POST	10	6	4	0
K_RECEIVE	7	7	0	0
K_RELEASE_PROCESS	49	18	9	22
K_SET_PROCESS_STATUS	6	4	2	0
K_SIGNAL	6	2	4	0
K_SPAWN	149	50	31	68
K_WALK_PROCESS_TABLE	2	2	0	0
(KER2)				
K_CLOSE	29	11	15	3
K_CREATE	29	10	1	18
K_GET_FILE_STATUS	28	0	6	22
K_LINK	30	3	3	24
K_MOUNT	66	28	28	10
K_OPEN	90	18	1	71
K_SECURE_TERMINAL_LOCK	7	4	0	3
K_SET_FILE_STATUS	81	15	15	51
K_UNLINK	59	3	4	52
K_UNMOUNT	79	22	19	38
(KER3)				
K_DEVICE_FUNCTION	100	49	51	0
K_SPECIAL_FUNCTION	3	2	1	0
K_WRITE_BLOCK	211	97	114	0
(KER4)				
K_BUILD_SEGMENT	39	22	3	14
K_GET_OBJECT_LEVEL	2	0	2	0
K_GET_SEGMENT_STATUS	3	0	3	0
K_RELEASE_SEGMENT	30	15	15	0
K_REMAP	82	55	27	0
K_RENDEZVOUS_SEGMENT	49	28	3	18
K_SET_OBJECT_LEVEL	11	5	0	6
K_SET_SEGMENT_STATUS	31	16	15	0
(KER5)				
K_READ_BLOCK	309	161	148	0
TOTAL	1827	743	563	521

KSOS Kernel Verification Results

2/12/80 Status of KSOS Specifications

(The total of the last three columns may be less than the first column, due to the formula generator eliminating duplicate formulas.)

	FOR	TRV	THM	UNP
(KER1)				
K_FORK	177	96	0	34
K_GET_PROCESS_STATUS	2	0	2	0
K_INTERRUPT_RETURN	9	9	0	0
K_INVOKE	61	42	11	1
K_NAP	0	0	0	0
K_POST	8	6	1	1
K_RECEIVE	7	7	0	0
K_RELEASE_PROCESS	74	19	5	13
K_SET_PROCESS_STATUS	6	4	2	0
K_SIGNAL	6	2	3	0
K_SPAWN	149	88	11	16
K_WALK_PROCESS_TABLE	2	2	0	0
(KER2)				
K_CLOSE	48	23	0	3
K_CREATE	26	12	1	6
K_GET_FILE_STATUS	7	0	3	0
K_LINK	15	11	2	1
K_MOUNT	64	23	6	17
K_OPEN	48	27	8	4
K_SECURE_TERMINAL_LOCK	7	4	0	2
K_SET_FILE_STATUS	36	16	10	1
K_UNLINK	26	19	3	1
K_UNMOUNT	79	35	6	17
(KER3)				
K_DEVICE_FUNCTION	45	13	32	0
K_SPECIAL_FUNCTION	3	2	1	0
K_WRITE_BLOCK	211	111	100	0
(KER4)				
K_BUILD_SEGMENT	43	33	3	3
K_GET_OBJECT_LEVEL	2	0	2	0
K_GET_SEGMENT_STATUS	3	0	2	0
K_RELEASE_SEGMENT	38	17	0	6
K_REMAP	50	34	9	7
K_RENDEZVOUS_SEGMENT	48	28	12	1
K_SET_OBJECT_LEVEL	11	7	0	4
K_SET_SEGMENT_STATUS	31	16	8	0
(KER5)				
K_READ_BLOCK	254	160	94	0
TOTAL	1590	867	337	138

KSOS Kernel Verification Results

2.2.1 Analysis of the Specification Proofs

The value of using an automatic tool for checking conformance with a formal model of security, rather than relying on careful scrutiny by teams of humans, became obvious when the tool detected numerous "errors" that had gone undetected throughout several iterations of human inspection by the Contractor, Subcontractor, and Customer. Analysis of these errors lead to the following three categorizations:

1. errors that could be removed by providing additional information or by syntactically reformulating the specifications;
2. errors that represent formal, but not "real", violations of the model;
3. errors that represent implicit channels, that cannot be removed without destroying needed KSOS functionality, but which can be bandwidth-limited;
4. and errors that represented wide-open security violations (there were no violations in this category).

Examples of errors that could be removed by adding or reformulating EXCEPTIONS are found in PROpost, PROreleaseProcess, PROsetprocessStatus, and FCAopen. Security violations existed in the original versions of the PRO specifications because a process could determine the (non)existence of another process at a higher level by means of an EXCEPTION value. In FCAopen, an error existed in the original specifications when a process tried to open a file at a higher level for writing (since file status information also had to be read by the process). The solution was to disallow a process from opening a file at a higher level.

An example of a violation that could be removed by kernel redesign was in the subtype mechanism. In FCA, a global assertion was needed stating that if a process p can read or write a file f, then p can read the subtype associated with f. Thus, the level of a file is greater than or equal to the level of its associated subtype. In the original design of subtypes, there was also the rule that to write on file f, a process p must be able to write on the subtype associated with f. All the previous facts, however, imply that for any subtype x, all files associated with x must be at the same level as x. This is unacceptable in the case of directories, since users at different levels create directories. The solution was to change the above rule so that to write on file f, process p must be able to execute (not write) the subtype associated with f.

An example of a formal, but not "real", security violation is a read reference to the openCount field of FCAinfo, as occurs in FCAClose. Ostensibly this read reference is a security violation; however, the only knowledge gained is whether or not its value is 1, and if so, the file is deleted immediately.

2.2.2 Additional Lessons Learned from the Kernel Specification Proofs

In addition to the direct analysis of the output of the specification proofs, certain principles emerged which have provided useful guidelines in

KSOS Kernel Verification Results

writing specifications, and which, in the future, may be worthwhile to incorporate in tools. Two such principles we have named the "setup principle" and the "transition principle".

2.2.2.1 The Setup Principle

The Setup Principle states: "If process p can access object o at time t , and no security-related changes occur to p or o between t and current time t' , then p can access o at t' without rechecking access rights". Applications of this principle are: putting a file into the open descriptor table; and putting a segment into an address space. Utilizing the principle in a proof would involve the restriction of tranquility violations to trusted software, and proving that trusted software did not make undesirable security-related changes.

2.2.2.2 The Transition Principle

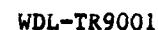
The Transition Principle states that "For finite, reusable objects (e.g., Seids, openDescriptors), tranquility violations are acceptable if they conform to the following transition rules:

1. in making transitions from a defined to an undefined state for an object o , all VFUNS whose security level is a function of o also become undefined at the same time (VFUNS such as SENseidNSP, whose security level is always system low, are thus excluded from this rule);
2. A new object comes into existence only from the undefined state."

2.3 Prototype Tools for Specification and Code Proofs

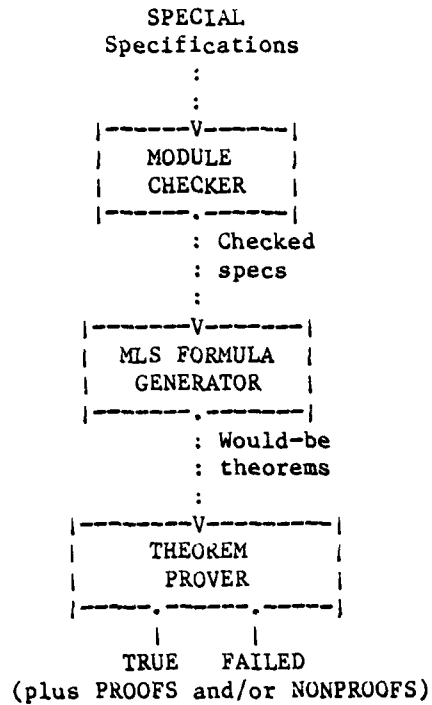
This section includes three figures showing the organization of the HDM tools, the organization of the MLS proof tools, and the organization of the code proof process (including a summary and some additional notes). These diagrams reflect the status of the subcontractor's efforts at the conclusion of their participation in KSOS methodological tool development, circa the first quarter of 1980.

Figure 1
ORGANIZATION OF THE HDM TOOLS



KSOS Kernel Verification Results

Figure 2
ORGANIZATION OF THE MLS SPEC PROOF TOOLS



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84

1000



22

KSOS Kernel Verification Results

2.3.1 Summary of Tools and Manual Steps

1. It is necessary before beginning the proof process that the specifications, mappings, and code conform to each other. Ideally this step is not an enormous undertaking if things are done consistently throughout. Problems that must be dealt with include various difficulties with exceptions, conflicting effects in multiple EFFECTS_OF, naming differences, different return argument conventions, etc. Special effort must also be devoted to handling sets and structures, and certain auxiliary VFUNs must also be introduced.

It should also be noted that this process is an iterative process. One of the most important aspects of this approach is that it detects inconsistencies. Thus each problem that is detected requires recycling through the appropriate paths in the figure.

2. MODULE CHECKER and MAPPING CHECKER are the HDM tools that check syntactic consistency. They have been working for four years, and are well documented.
3. Manual translation to CIS (Common Internal Specification) and CIM (Common Internal Mapping) removes all quantification, accommodates structures (rewriting VFUN references in terms of SELECT, UPDATE, and MAKESTRUCT), expands nested macros, etc.
4. "UPDATE" EXPANDER translates CIS and CIM to the internal specification form (VSSL) used by the verification system. It removes all uses of UPDATE and expands them to expressions in terms of SELECT only — guaranteeing consistent manipulations of structures. Except for the structure representations, CIS/CIM and VSSL are identical.
5. CODE TO PIF PARSER parses the Modula code into a parsed internal form. This form is normally invisible to the prover.
6. PIF TO CIF TRANSLATOR translates the parsed internal form into the common internal code form used by the verification system, using the supplied (upper-level) specifications to compute exception handling instructions. CIF is documented in the Boyer-Moore HDM document.
7. Insertion of various specifications (for the upper-level SELECT, [called SELECT1], the lower-level SELECT, IS.STRUCTURE, CLOCKINFO, and the notion of UNDEFINED) is required to complete the specifications.
8. The CIF must be augmented with the mappings, CLOCKINFO, at least two definitions, an invariant (which may contain many components), global variables (which are optional), and an initialization program.
9. The VERIFICATION CONDITION GENERATOR takes upper- and lower-level specifications and code (augmented with the mappings, etc., as noted above), and generates verification conditions for the Theorem Prover.
10. The THEOREM PROVER takes verification conditions as would-be theorems and attempts to prove them. It returns either TRUE (along with the proof) or

KSOS Kernel Verification Results

FAILED (along with its attempted proof) for each verification condition.

2.3.2 Additional Notes

- A. Before attempting any code proofs, code should have been compiled, run, and tested. These steps are omitted from this diagram.
- B. This path may be traversed either manually or automatically (in the latter case, with the ALLPARSE function).
- C. The output from the CIF translator must be manually loaded from the translator environment into the Theorem Prover environment, which are disjoint. However, this change of environment could be automated.
- D. At present the Theorem Prover must be invoked manually for the given set of verification conditions, although this could easily be done automatically.

2.3.3 Sample MLS Tool Outputs and Their Interpretation

In this section we present two sample outputs from the MLS tool: one which fails to be proven subsequently by the theorem prover, and one which is subsequently proven by the theorem prover.

2.3.3.1 A Failed Formula

This example is taken from PVMbuild, and is generated from the first effect. The relevant definitions are the following:

DEFINITIONS

```
tiiStruct proTii IS TIIinfo(pSeid);
tiiStruct segTii
  IS STRUCT(proTii.nd, ... );
seid newSegSeid
  IS SOME seid s | SENseid Nsp(s) = SENseidNsp(exampleSegmentSeid)
                  AND SEGinstanceInfo(s) = ?;
```

EXCEPTIONS

...

EFFECTS

```
'TIIinfo(newSegSeid) = segTii;
...;
```

The MLS tool generates the following conjecture from this first effect, which is then fed into the theorem prover:

KSOS Kernel Verification Results

Proving:
(SMXcompare pSeid.1 s.1.1.1.1)

Name the conjecture *1.

Since there is nothing to induct upon, the proof has

F A I L E D !

The conjecture is generated based on the structural aspect of the effect, namely, that writing is occurring into TIIinfo, as evidenced by the syntax of TIIinfo being quoted. Hence, according to the multilevel security model, which allows writing to occur only in an upward or equal direction of security level, the model requires the source of the write to be less than or equal to the level of 'TIIinfo. The predicate "x is less than or equal to y in security level" is given formally by "SMXcompare(x,y)." The source of the writing is segTii, which expands from the above definitions to STRUCT(TIIinfo(pSeid), ...). The level of segTii is computed by the MLS tool to be pSeid, based on information fed to the tool at the start of the session (or remembered by the tool from a previous session). The level of TIIinfo(newSegSeid) is likewise computed by the MLS tool to be s (from the definition of newSegSeid). Hence the model requires pSeid to be less than or equal to s in security level, and generates the conjecture so stating.

The reason for the arguments of the conjecture being named pSeid.1 and s.1.1.1.1 are due to the internal workings of the MLS tool, which adds sufficient "tails" to the variable names to make them all unique. The MLS tool works in a completely flat name space.

For the above conjecture to be true, it is necessary to add more information, namely, that the level of newSegSeid is greater than or equal to the level of pSeid. This additional information can be added in a variety of ways. One way would be to add a third conjunct to the definition of newSegSeid. The information could also be added via an exception.

2.3.3.2 A Proven Formula

Virtually all of the formulas generated by the MLS tool and subsequently proved were simple in nature, requiring substitution of variables, or propositional logic, rather than complex inductive strategies. The following example is typical.

As a first step, a global assertion stating that

SMXcompare(SEGuseInfo(s, sd).instance, s) = TRUE

was added to the semantics of the system, taken from an assertion in the specifications. Universal quantification is understood with respect to the parameters s and sd. The internal form of the assertion is:

KSOS Kernel Verification Results

```
-ADD.AXIOM(A0013 (REWRITE)
              (IMPLIES T
                (EQUAL (SMXcompare (DOT (SEGuseInfo sl sd)
                                         (QUOTE instance))
                                   sl)
                      T))
              NIL)
```

The following proof is now presented, which uses the above axiom.

```
Module: ker4
Function: K_build_segment
EXCEPTION
(EXCEPTIONS_OF (PVMbuild pSeid ss ms size vl))

Proving:
(IMPLIES (EQUAL use.1.1.1.1 (SEGuseInfo pSeid.1 s.4.1))
         (SMXcompare (DOT use.1.1.1.1 (QUOTE instance))
                     pSeid.1))
```

This formula simplifies, rewriting with A0013, to:

(TRUE).

2.4 Code Proof for a Simplified Module

The Modula procedure SMXcompare was greatly simplified for the purpose of pushing a code proof through the tools. Essentially all data abstractions were removed from the procedure and its associated specifications, and the procedure simply compared the fields of the two objects under consideration for the appropriate inequality or subset operation. Although very much a toy example, it was instructive to see the trace of the theorem prover in proving the "correctness of the implementation of SMXCompareModule on PrimitiveModule". The elapsed time was 1.065 seconds, with .124 seconds of cpu time devoted to theorem proving. This code proof has been included as Appendix B.

2.5 Manual Code/Specification Analysis

As part of the efforts leading to the final version of the kernel specifications, we spent approximately 2 man days in manually comparing the code and specifications for the PVM module. Our goal was to get a feeling for the difficulties in carrying out a code proof. The first step was to map the types between the code and the specifications. Although there were no conceptual difficulties at this step, a variety of minor issues had to be checked. For example, {segDes} is mapped into {0..15}. We checked that no ordering properties of the integers were used at the top level in the code (where such ordering properties would be unavailable at the top level in the specifications). It was only in the implementation of the FORALL construct in SPECIAL that the ordering properties of the integers were used. Another minor example occurs in the mapping of tiiStruct. Although the mapping is virtually the identity mapping, in the specifications, owner and group are of type INTEGER and in the code they are of type CARDINAL.

KSOS Kernel Verification Results

The next step was to map the primitive VFUNS into the Modula data representations. There are five primitive VFUNS to be dealt with. Four of them had reasonably clean mappings. `SEginUseIndexSet`, on the other hand, had no counterpart in the code. We conjectured that its use in the specifications was to guarantee unique `seid` generation in `PVMbuild` and `PVMcopySeg`. In the code, `STMGassignEntry` has the property that any `seid` generated is unequal to the `seid` of any existing segment. It was unclear how a mechanical code proof would account for this type of correspondence.

The next step was to map the definitions between the specifications and the code. We noted details in the code that had no counterpart in the specifications, e.g.: all virtual addresses that are base addresses for upward-growing segments are multiples of 64; no segment size exceeds $2^{16} - 512$; and all segment sizes are multiples of 512. Nevertheless, there were no major conceptual problems in this mapping.

The final step was to map the OFUNS into the Modula procedures. Our approach was to attempt "transliterating" the OFUNS into a Modula-like pseudo-code by first applying the previous mappings (and thus moving into a "Modula-like data space"), and then applying programmers's license to convert the nonprocedural aspects of `SPECIAL` into the standard sequential type of Modula procedure. Having done this, the goal was then to compare the resulting pseudo-code with actual Modula code to see if we could demonstrate equivalence. This paradigm went smoothly for the first OFUN we attempted (`PVMcreate`). For the next OFUN, however, (`PVMstore`) we encountered conceptual obstacles. In the specifications, one of the parameters was `VECTOR_OF_INTEGER vec`; in the code, however, there was nothing tangible corresponding to this because `vec` represents data in transit along i/o channels or the data bus. To achieve a code proof would have necessitated resolving this problem, e.g., by formalizing appropriate aspects of the underlying computer architecture and incorporating them into the specifications. This was obviously undoable at this late stage.

KSOS Kernel Verification Results

3. Non-Achievements

As mentioned earlier, there were various goals that were not achieved. In this section we indicate difficulties which made success elusive, and where appropriate, indicate things we would do differently which might lead to greater success in the future.

1. Logistical difficulties in running verification. Getting access to the verification machine at SRI involved physically being at SRI (due to a poor communications link between FACC and SRI). Furthermore the jobs had to be run in batch mode in the evening hours. In the future, things could be improved dramatically with an on-site, accessible, interactive verification capability.
2. Tool development in isolation from applications development. A large part of the SRI subcontract for tool development and related support proceeded in virtual isolation from the KSOS development effort. In the future, a much closer relationship is called for between Contractor and Subcontractor. The design and development of tools should be coordinated with the applications that will eventually use the tools.
3. Failure to apply HDM throughout all stages of KSOS. Although the modularization achieved by the formal specifications is clean and comprehensible, there was a failure to apply the hierarchical aspects of the methodology. The original lower-level specifications were simply transliterations from the Modula code to SPECIAL, rather than an evolution from top-level specifications. The gap between upper- and lower-level specifications was a significant reason for failure to achieve code proofs. In the future, more rigorous use of all aspects of HDM will be required. In particular, implementation should not commence until there is a clear hierarchy of specifications, with appropriate mapping functions between adjacent levels, in which there is a reasonable gap between the bottom level and the state space corresponding to the implementation language. (It is worth noting that FACC has applied HDM in its entirety, as just mentioned, in IR&D projects during 1980 with successful results).
4. Shortcomings of Specification Technology. The lack of powerful constructs in SPECIAL (and all other specification languages) for concurrency and dynamic processes created another gap between the code and the specifications. For future applications, more powerful versions of HDM, in which SPECIAL is buttressed by appropriate concurrency and process constructs, and in which there is a closer correlation between HDM and the underlying theorem prover, will help achieve verification goals.
5. Oversimplifications in the basic security model. The Bell and LaPadula model which formed the basis for the KSOS security model is inappropriate in several respects, e.g., there is no direct way to model KSOS privileges (which is a major reason the NKSIR specifications were not proven), and it does not allow the reuse of resources such as seeds (which is required in a finite implementation). In the future, more sophisticated and relevant models should be developed.

KSOS Kernel Verification Results

4. Appendix A - Kernel Formal Specifications

This appendix contains the most recent version of the Kernel Formal Specifications that were actually verified. The current version, which describes the system actually delivered, appears as an appendix to the Kernel B-Specs.

```

1  $(" MODULE:      fca.specs (version 2.21)
2  CONTENTS:      File Capabilities
3  TYPE:          SPECIAL.specifications
4  LAST CHANGED:  10/12/79, 11:26:25
5  ")
6
7
8  MODULE fca
9
10
11 $("This module manages all openable objects, i.e., those that are referenced
12 through the open table corresponding to a process. These objects include
13 files, devices - both addressable and nonaddressable-, terminals, extents,
14 and subtypes.
15
16 Each object is identified by a seid. Seids for devices, terminals, and
17 subtypes are allocated at system generation time. These objects are
18 permanent, and cannot be dynamically allocated and deallocated.
19 Seids for files are allocated by this module. Seids for extents are
20 allocated when the device is physically mounted. Physical mounting
21 is not handled at this time - logical mounting is - but should be.
22
23 Each process at creation is assigned an open table, in which all the
24 open objects of that process are recorded, along with their mode of
25 access. The state of the open table for a process is recorded in the
26 values of the V-functions 'FCAopenTableExists(pSeid)' which tells
27 whether the open table for the process named by 'pSeid' exists, and
28 'FCAopenEntry(pSeid, od)', which gives the seid and open mode for the
29 open object of process 'pSeid' named by the open Descriptor - a
30 designator - 'od.'
31
32 The existence of an openable object is detected by a defined value for
33 'FCAfileStatusInfo(fSeid)', where 'fSeid' is the object's seid. Each
34 object's type is ascertained by looking at the nsp part of the seid.
35 Depending on the type of the object, certain V-functions hold additional
36 information. A description of this information can be found in the
37 comment directed at each type of object.")
38
39 $(" DEVICES -- there are two kinds of devices, addressable and nonaddressable;
40 an addressable device, such as a disk, has two properties: it can be
41 accessed via a block number or address; and what is put onto the device
42 via a read operation is retrieved by a write when the device is read at
43 the same address. A non-addressable device, such as a tape unit, can be
44 viewed as having an infinite stream of input data and producing an
45 infinite stream of output data. Each kind of device has four quantities
46 associated with it: a minimum request, a maximum request, a size
47 modulus, and a maximum block number. For non-addressable devices, the
48 size modulus must be 1 and the maximum block number must be zero.
49 An IO request specifies a certain number of characters at a certain
50 block number. The number of characters must be within the range
51 defined by the minimum and maximum request quantities for the device,
52 and must be a multiple of the size modulus. The block number must be with
53 within the range {0 .. maximum block number} for the device.
54 ")
55
56 $(" TERMINALS -- With one exception, terminals are nonaddressable devices

```

57 whose IO requests are limited to small multiples of a single character
 58 (~ 255). Terminals however, have a special property. To enable the
 59 user to change the security level of his job without changing terminals,
 60 there is an illusion that a single terminal is represented by a
 61 multiplicity of device seids, one for each possible login security level.
 62 Each seid represents a particular secure path to the terminal. Only
 63 one path to the terminal may do IO operations at a time, and this path
 64 is specified by the value of the V-function 'FCACurrentPath(t)', where
 65 t refers to the terminal group or physical terminal.
 66 The different paths associated with a particular physical terminal
 67 are specified by the value of the V-function 'FCATerminalPathSet(t)',
 68 where t is as above."
 69

70 \$(" EXTENTS — Extents are addressable devices, representing areas on a
 71 disk, with a block size of 512 and a size determined when the device is
 72 physically mounted. For the
 73 purposes of this specification, the size has been predetermined, as no
 74 physical mounting is specified. The special property of extents is that
 75 they can be logically mounted, so that they 'become' a file system
 76 or set of files. When
 77 the system is started up, there are no files, except the root, only
 78 extents. Each extent is mounted, setting up the actual file system
 79 that a user sees when he logs on. When the extent is mounted, it can
 80 no longer be accessed as an extent. Unmounting turns a file system
 81 back into an extent, and the file system disappears."
 82

83 \$(" FILES — Files are addressable devices, with a block size of 512 and
 84 two important properties. They can be dynamically created and deleted,
 85 and they are of variable size. Writing onto the end of a file effectively
 86 changes its size. Files may also be linked to. A link is a reference
 87 count used by the directory manager sitting above the kernel. It
 88 represents the number of directories in which a file is found. When
 89 this count goes to 0, and no process has the file open, the file
 90 is deleted. This is the only way of deleting files, although they can
 91 be explicitly created."
 92

93 \$(" SUBTYPES — This is an additional protection mechanism over and above
 94 that provided by the mandatory, privilege, and discretionary access
 95 control systems. Each openable object may be associated with a
 96 subtype, of which there are a fixed number at system generation time.
 97 Any object of a non-null subtype may be accessed only by those processes
 98 who have access rights to the subtype as well as the object. The
 99 access right to a subtype is established by opening the subtype for
 100 the desired access. Access to the subtype is granted or denied according
 101 to the usual mandatory and discretionary rules. An object with
 102 a non-null subtype can be accessed only when the open descriptor for
 103 the subtype, to which access must already have been granted, is
 104 presented, and when the desired access to the object is a subset of
 105 the access granted to the subtype. This forms a mini-capability
 106 mechanism with type extension, which is necessary for achieving access
 107 control over objects such as directories. Thus there will be a
 108 directory subtype, to prevent arbitrary programs from damaging the
 109 directory system just because they have access to a particular
 110 directory. The rules for subtype access occur in the open function and
 111 all functions that require a subtype capability for accessing
 112 an object.")

```

113
114
115     TYPES
116
117     $(FROM smx)
118 nonDisType: STRUCT_OF(
119         INTEGER securityLevel; SET_OF securityCat securityCatS;
120         INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
121 daType: SET_OF daMode;
122 modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
123 tiiStruct: STRUCT_OF(nonDisType nd; modeStruct da; INTEGER owner, group;
124                     SET_OF privType priv);
125
126     $(from fca -- exportable)
127 openDescriptor: DESIGNATOR;
128 openModes: {omRead, omWrite, omExclusive};
129 IOfunction: {rewind, etc}; $(names for special kinds of IO functions)
130 deviceType: {RK05, RWPO4, RWPO5, RWPO6, RSW04, TWE16, TM11, TU56, PR11,
131             PC11, LP11, IMP11B, LHDH};
132 terminalGroup: DESIGNATOR;
133
134     $(from fca -- redeclarable)
135 fileStatus: STRUCT_OF(INTEGER nBlocks, linkCount, timeLastMod; seid subtype;
136                     BOOLEAN openAtCrash);
137     $(data about an openable object that is returned to the user)
138 globalData: STRUCT_OF(INTEGER linkCount, timeLastMod; seid subtype;
139                     BOOLEAN openAtCrash);
140     $(state information for all openable objects)
141 fileBlock: VECTOR_OF CHAR;
142 ioStatus: STRUCT_OF(INTEGER devIndep, devDep);
143     $(result of an IO operation, including possible hardware failure)
144 openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);
145     $(entry in a process' open file table)
146 asyncId: CHAR;
147 readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);
148 deviceStruct: STRUCT_OF(BOOLEAN addressable;
149                     INTEGER minRequest, maxRequest, modSize, maxBlockNo);
150     $(properties necessary for processing IO requests for devices)
151 mountTableEntry: STRUCT_OF(seid leafSeid, rootSeid; BOOLEAN readOnly;
152                     tiiStruct devTii; globalData devGl);
153 fileSystemEntry: STRUCT_OF(seid fileSeid; globalData gl; tiiStruct tii;
154                     VECTOR_OF fileBlock fileData);
155     $(the state of a file, for purposes of mounting and unmounting)
156 fileSystem: SET_OF fileSystemEntry;
157     $(the state of an entire mountable file system)
158 sodPair: STRUCT_OF(seid ps; openDescriptor od);
159     $(for openCount definition)
160
161
162     PARAMETERS
163
164 SET_OF seid subtypeSeidSet; $( set of non-null subtypes known to the system)
165 seid FCAstSeid; $(all objects who have this seid as their subtype ARE
166     in fact subtypes)
167 seid nullStSeid; $( seid indicating the null subtype)
168 openDescriptor FCAnullSt; $(a file descriptor indicating the null subtype)

```

```

169 seid FCARootSeid; $(distinguished root of KSOS permanent file system)
170 INTEGER FCAMaxOpenDescriptors; $(maximum number of open descriptors per
171 process)
172
173
174 DEFINITIONS
175
176 INTEGER FCAfileSize(seid fSeid) IS
177 CARDINALITY({INTEGER i | FCAfileData(fSeid, i) ~= ?});
178 $(the size, in blocks, of an addressable device, file, or extent)
179
180 INTEGER nOpenDescriptors(seid pSeid) IS
181 CARDINALITY({openDescriptor od | FCAopenEntry(pSeid, od) ~= ?});
182 $(the number of open objects in a given process: it must not exceed a fixed
183 maximum)
184
185 INTEGER openCount(seid fSeid) IS
186 CARDINALITY({sodPair sp | FCAopenEntry(sp.ps, sp.od).openSeid = fSeid});
187 $(the number of times that a given openable object is open)
188
189 deviceStruct deviceDataSeid(seid fSeid) IS
190 FCAdeviceData(FCAdeviceType(fSeid));
191 $(given the seid of a device, the data on which its IO requests depend)
192
193 SET OF daMode h_modeTrans(SET OF openModes oModes) IS
194 (IF omRead INSET oModes THEN {daRead} ELSE {})
195 UNION (IF omWrite INSET oModes THEN {daWrite} ELSE {});
196 $(translates from one enumerated type, open Modes, to another slightly
197 different enumerated type, discretionary access modes)
198
199 BOOLEAN isCurrentPath(seid tSeid) IS
200 EXISTS terminalGroup t : FCAcurrentPath(t) = tSeid;
201 $(for a given terminal, tells whether it is the active path to its physical
202 device)
203
204 BOOLEAN isReadOnly(seid s) IS
205 EXISTS seid devSeid
206   SENSEidNsp(FCAmountTable(devSeid).rootSeid) = SENSEidNsp(s)
207   AND FCAmountTable(devSeid).readOnly = TRUE;
208 $(tells whether or not a given file is on a file system that is mounted in
209 read-only mode)
210
211 seid indir(seid fSeid) IS
212 LET seid devSeid | FCAmountTable(devSeid).leafSeid = fSeid
213 IN IF devSeid = ? THEN fSeid ELSE FCAmountTable(devSeid).rootSeid;
214 $(if a file represents the spot in the file system where a file system
215 has been mounted, it is translated into the root of the mounted file
216 system)
217
218 EXTERNALREFS
219
220 FROM mac:
221 VFUN MACclock() -> INTEGER time;
222
223 FROM smx:
224 seid: DESIGNATOR:

```

```

225 secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
226                     tExtent, tNull};
227 privType: {
228     privFileUpdateStatus,    privLink,        privLockSeg,
229     privModifyPriv,          privMount,
230     privSetFileLevel,        privSetSegProcLevel,
231     privStickySeg, privTerminalLock,
232     privViolSimpSecurity,    privViolStarSecurity,
233     privViolSimpIntegrity,  privViolStarIntegrity,
234     privViolDiscrAccess,    privSignal,      privWalkPTable,
235     privHalt,                privKenrelCall, privViolCompartments,
236     privRealizeExecPermissions};
237 daMode: {daRead, daWrite, daExecute};
238 securityCat: DESIGNATOR;
239 integrityCat: DESIGNATOR;
240 VFUN SENseidNsp(seid s) -> INTEGER nsp;
241 VFUN SENseidType(seid s) -> secureEntityType set;
242 VFUN TIIinfo(seid s) -> tiiStruct tiist;
243 VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
244 VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;
245 VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
246
247
248     ASSERTIONS
249
250 FORALL seid fSeid | FCAinfo(fSeid) ~= ?
251   : SENseidType(fSeid) INSET {tTerminal, tDevice, tSubtype, tFile, tExtent};
252   $(restricts the types of objects manipulated by this module)
253
254 FORALL seid fSeid: {INTEGER i | FCAfileData(fSeid, i) ~= ?}
255   = {0 .. FCAfileSize(fSeid) - 1};
256   $(the blocks of a file, extent, or addressable device form a sequence)
257
258 FORALL seid dSeid
259   | SENseidType(dSeid) = tDevice AND FCAinfo(dSeid) ~= ?
260   : (LET deviceStruct d = deviceDataSeid(dSeid)
261     IN NOT d.addressable => d.modSize = 1 AND d.maxBlockNo = 0);
262   $(necessary properties of all non-addressable devices)
263
264 FORALL seid fSeid
265   | FCAinfo(fSeid) ~= ?
266   : (LET secureEntityType t = SENseidType(fSeid)
267     IN FORALL INTEGER i | FCAfileData(fSeid, i) ~= ?
268       : LENGTH(FCAfileData(fSeid, i))
269         = (IF t = tDevice THEN deviceDataSeid(fSeid).modSize
270           ELSE IF t INSET {tExtent, tFile} THEN 512
271           ELSE ?));
272   $(the lengths of all blocks for files, extents, or addressable devices
273     must correspond to the parameters for those objects)
274
275 FORALL seid s | SENseidType(s) = tTerminal AND FCAinfo(s) ~= ?
276   : (EXISTS terminalGroup t1
277     : s INSET FCAterminalPathSet(t1)
278       AND (FORALL terminalGroup t2 ~= t1
279         : NOT s INSET FCAterminalPathSet(t2)));
280   $(each terminal is in exactly one terminal group)

```

```

281
282 FORALL seid f
283   | EXISTS INTEGER i : FCAfileData(f, i) ~= ?
284   | FCAinfo(f) ~= ?
285   AND (SENseidType(f) = tDevice AND deviceDataSeid(f).addressable = TRUE
286       OR SENseidType(f) INSET {tFile, tExtent});
287   $(only files, extents, or addressable devices have file data)
288
289 FORALL seid f
290   : (FCAinputStream(f) ~= ? AND FCAoutputStream(f) ~= ?)
291   = (SENseidType(f) = tTerminal
292     OR SENseidType(f) = tDevice
293     AND deviceDataSeid(f).addressable = FALSE);
294   $(non-addressable device have both an input and an output stream, although
295     it may always be null)
296
297 FORALL seid pSeid | FCAopenTableExists(pSeid)
298   : FCAopenEntry(pSeid, FCAnullSt) = ?;
299   $(there are never any opened objects assigned to the open descriptor
300     reserved for the null subtype)
301
302 FORALL seid s1, s2
303   : SENseidType(s1) = tSubtype AND SENseidType(s2) = tSubtype
304   => SENseidNsp(s1) = SENseidNsp(s2);
305   $(subtypes have only one name space partition)
306
307 SENseidType(nullStSeid) = tSubtype AND SENseidType(FCastSeid) = tSubtype;
308   $(the null subtype seid and the subtype seid indicating "sybtype")
309
310 FORALL seid s INSET subtypeSeidSet : SENseidType(s) = tSubtype;
311   $(properties of the set of non-null subtypes)
312
313 SENseidType(FCARootSeid) = tFile;
314   $(property of the distinguished root of the entire file system)
315
316 FORALL seid f | FCAinfo(f) ~= ? AND SENseidType(f) = tFile
317   : EXISTS seid d : SENseidNsp(FCAMountTable(d).rootSeid) = SENseidNsp(f)
318   OR f = FCARootSeid;
319   $(a file is either the root or it is on a mountable file system)
320
321
322 FUNCTIONS
323
324 $(----- state functions --- devices -----)
325
326 VFUN FCADeviceType(seid fSeid) -> deviceType d;          $(FCADeviceType)
327   $(gives the type of a particular device, which determines its IO behavior)
328   HIDDEN;
329   INITIALLY
330     (d ~= ?)
331     = (SENseidType(fSeid) = tDevice AND FCAinfo(fSeid) ~= ?);
332
333 VFUN FCADeviceData(deviceType d) -> deviceStruct ds;      $(FCADeviceData)
334   $(for a given type of device, defines the IO behavior)
335   HIDDEN;
336   INITIALLY

```

```

337 ds = (IF d = RK05
338     THEN STRUCT(TRUE, 512, 32768, 512, 203*24-1)
339     ELSE IF d INSET {RWPO4, RWPO5}
340     THEN STRUCT(TRUE, 512, 32768, 512, 22*19*411-1)
341     ELSE IF d = RWPO6
342     THEN STRUCT(TRUE, 512, 32768, 512, 22*19*411*2-1)
343     ELSE IF d = RSWO4 THEN STRUCT(TRUE, 512, 32768, 512, 2048-1)
344     ELSE IF d INSET {TWE16, TM11} THEN STRUCT(FALSE, 12, 8191, 1, 0)
345     ELSE IF d = TU56 THEN STRUCT(TRUE, 512, 512, 512, 0)
346     ELSE IF d INSET {PR11, PC11} THEN STRUCT(FALSE, 1, 1, 1, 0)
347     ELSE IF d = LP11 THEN STRUCT(FALSE, 1, 132, 1, 0)
348     ELSE IF d INSET {IMP11B, LHDH} THEN STRUCT(FALSE, 1, 8191, 1, 0)
349     ELSE ?);
350
351 $(----- state functions -- terminals -----)
352
353 VFUN FCATerminalPathSet(terminalGroup t) -> SET_OF seid ss;
354                                     $(FCATerminalPathSet)
355 $(defines the set of paths, or logical terminals, that correspond to
356  a given terminal group, or physical terminal)
357 HIDDEN;
358 INITIALLY
359     FORALL seid s INSET ss
360     : FCAinfo(s) ~= ? AND SENSEidType(s) = tTerminal;
361
362 VFUN FCAcurrentPath(terminalGroup t) -> seid s; $(FCAcurrentPath)
363 $(the seid of the logical terminal that is allowed to do IO on a given
364  physical terminal)
365 HIDDEN;
366 INITIALLY
367     s INSET FCATerminalPathSet(t);
368
369 $(----- state functions -- mountable file systems -----)
370
371 VFUN FCAMountTable(seid extentSeid) -> mountTableEntry mte; $(FCAMountTable)
372 $(for a given extent that is mounted, tells leaf of the old file system,
373  the root of the new or mounted file system, whether the file system
374  is read only, and the state information for the extent)
375 HIDDEN;
376 INITIALLY mte = ?;
377
378 VFUN FCAextentToFileSys(VECTOR_OF fileBlock fb; seid rootSeid)
379 -> fileSystem fs; $(FCAextentToFileSys)
380 $(given the data of a given extent and a root seid for a file system
381  to be made out of the extent whose data is given, produces the
382  file system consisting of a set of tuples each made up of a
383  file seid and the state of the file)
384 HIDDEN;
385 INITIALLY
386     fs ~= ? =>
387     (EXISTS fileSystemEntry fse INSET fs : rootSeid = fse.fileSeid)
388     AND (FORALL fileSystemEntry fse INSET fs
389         : SENSEidNap(fse.fileSeid) = SENSEidNap(rootSeid));
390     $(a constant function for data conversion)
391
392 $(----- state functions -- all openable objects -----)

```

```

393
394 VFUN FCAinfo(seid fSeid) -> globalData gl; $(FCAinfo)
395 $(the status information, excluding data and type dependent stuff, for
396 an openable object)
397 HIDDEN;
398 INITIALLY
399 IF deviceDataSeid(fSeid) ~= ? THEN gl ~= ?
400 ELSE IF fSeid INSET {FCARootSeid, FCAstSeid, nullStSeid}
401 THEN gl.subtype = nullStSeid
402 ELSE IF fSeid INSET subtypeSeidSet THEN gl.subtype = FCAstSeid
403 ELSE gl = ?;
404
405 VFUN FCAfileData(seid fSeid; INTEGER blockNo) -> fileBlock fb; $(FCAfileData)
406 $(the data contained on a file, extent or an addressable device)
407 DEFINITIONS
408 secureEntityType type IS SENseidType(fSeid);
409 deviceStruct d
410 IS IF type = tDevice THEN deviceDataSeid(fSeid)
411 ELSE IF type INSET {tFile, tExtent}
412 THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid)-1)
413 ELSE STRUCT(FALSE, 1, 255, 1, 0);
414 HIDDEN;
415 INITIALLY
416 (IF FCAinfo(fSeid) = ? OR type INSET {tTerminal, tSubtype}
417 OR NOT blockNo INSET {0 .. d.maxBlockNo} OR NOT d.addressable
418 THEN fb = ?
419 ELSE fb ~= ?)
420 AND (fb ~= ?
421 => LENGTH(fb) = d.modSize
422 AND (FORALL INTEGER i INSET {0 .. blockNo - 1}
423 : FCAfileData(fSeid, i) ~= ?)
424 AND (FORALL INTEGER j INSET {1 .. LENGTH(fb)}
425 : fb[j] ~= ?));
426
427 VFUN FCAinputStream(seid fSeid) -> VECTOR_OF_CHAR vc; $(FCAinputStream)
428 $(the input data for terminals and nonaddressable devices)
429 HIDDEN;
430 INITIALLY
431 IF FCAinfo(fSeid) ~= ?
432 AND (SENseidType(fSeid) = tTerminal
433 OR SENseidType(fSeid) = tDevice
434 AND deviceDataSeid(fSeid).addressable = TRUE)
435 THEN vc ~= ?
436 ELSE vc = ?;
437
438 VFUN FCAoutputStream(seid fSeid) -> VECTOR_OF_CHAR vc; $(FCAoutputStream)
439 $(the output data for terminals and nonaddressable devices)
440 HIDDEN;
441 INITIALLY
442 IF FCAinfo(fSeid) ~= ?
443 AND (SENseidType(fSeid) = tTerminal
444 OR SENseidType(fSeid) = tDevice
445 AND deviceDataSeid(fSeid).addressable = TRUE)
446 THEN vc ~= ?
447 ELSE vc = ?;
448

```

```

449 $(----- state functions --- open tables -----)
450
451 VFUN FCAopenTableExists(seid pSeid) -> BOOLEAN b:      $(FCAopenTableExists)
452   $(the existence predicate for the table of openable objects corresponding
453     to a given process)
454   HIDDEN:
455   INITIALLY b = FALSE;
456
457 VFUN FCAopenEntry(seid pSeid; openDescriptor od) -> openFileEntry oe;
458   $(the information in entry "od" of the open table of process "pSeid")
459   HIDDEN:
460   INITIALLY oe = ?;
461
462 $(----- creation of files -----)
463
464 OVFUN FCAcreate(seid pSeid, nspSeid; modeStruct da; openDescriptor stCap)
465   -> STRUCT_OF(seid fSeid; openDescriptor od) return:      $(FCAcreate)
466   $(creates a file and opens it in the desired mode.  the file system on
467     which the file resides must be writable.  If a subtype capability
468     is provided it must refer to a valid subtype.  The created file gets
469     only the discretionary access permission specified.  All other data
470     comes from the process making the call.  The file is originally of zero
471     length)
472   DEFINITIONS
473     tiiStruct ptii IS TIIinfo(pSeid);
474     tiiStruct otii IS STRUCT(ptii.nd, da, ptii.owner, ptii.group, ptii.priv);
475     seid extSeid
476       IS SOME seid s | SENseidNsp(FCAmountTable(s).rootSeid)
477         = SENseidNsp(nspSeid);
478   EXCEPTIONS
479     $(these exceptions subsume those for opening a file for writing)
480     KefcaBadNsp:
481       extSeid = ? OR NOT SMXflow(pSeid, extSeid, {daRead, daWrite});
482     KefcaNoWriteDa: NOT daWrite INSET da.ownerMode;
483     KefcaDevNotWritable: isReadOnly(nspSeid);
484     KefcaBadSubtype: stCap ~= FCAnullSt
485       AND FCAinfo(FCAopenEntry(pSeid, stCap).openSeid).subtype ~= FCAstSeid;
486     KefcaSTNotWritable:
487       stCap ~= FCAnullSt
488       AND NOT omWrite INSET FCAopenEntry(pSeid, stCap).openMode;
489     KefcaOdSpace: nOpenDescriptors(pSeid) >= FCAMaxOpenDescriptors;
490     RESOURCE ERROR;
491   ASSERTIONS
492     FCAopenTableExists(pSeid);
493   EFFECTS
494     LET seid fs | FCAinfo(fs) = ?;
495     openDescriptor o | FCAopenEntry(pSeid, o) = ? AND o ~= FCAnullSt
496     IN return = STRUCT(fs, o)
497       AND 'TIIinfo(fs) = otii
498       AND 'FCAopenEntry(pSeid, o) = STRUCT(fs, {omWrite})
499       AND 'FCAinfo(fs) = STRUCT(0, MACclock().
500                                     IF stCap = FCAnullSt THEN nullStSeid
501                                     ELSE FCAopenEntry(pSeid, stCap).openSeid,
502                                     FALSE);
503
504 $(----- file opening and closing -----)

```

```

505
506 OVFUN FCAopen(seid pSeid, oSeid; SET_OF openModes mode; openDescriptor stCap)
507     -> openDescriptor od;                                     $(FCAopen)
508     $(opens the openable object specified by "oSeid". The object must exist
509     and the process "pSeid" must have the right mandatory and discretionary
510     access. Special rules, described below, apply to subtypes.
511     Special rules also apply when the object is being opened for exclusive
512     use. Only one process may open an object for exclusive use.
513     A process is allowed a fixed maximum number of open objects.)
514 DEFINITIONS
515     seid o IS indir(oSeid);
516     globalData ofst IS FCAinfo(o);
517     openFileEntry stEntry IS FCAopenEntry(pSeid, stCap);
518     BOOLEAN anotherExcl IS
519         EXISTS seid pSeidl; openDescriptor odl:
520             FCAopenEntry(pSeidl, odl).openSeid = o AND
521             omExclusive INSET FCAopenEntry(pSeidl, odl).openMode;
522     $(the object is opened exclusive use elsewhere)
523 EXCEPTIONS
524     KefcaNoFile: ofst = ? OR NOT SMXflow(pSeid, o, h_modeTrans(mode));
525     KefcaBadRefCount: ofst.linkCount = 0;
526     KefcaBadStCap:
527         stCap ~= FCAnullSt AND FCAinfo(stEntry.openSeid).subtype ~= FCastSeid:
528         $(a subtype capability was specified, but it is not for a valid subtype)
529     KefcaNoStCap:
530         stCap = FCAnullSt AND NOT ofst.subtype INSET {nullStSeid, FCastSeid};
531         $(no subtype capability was specified, but the object has a non-null
532         subtype)
533     KefcaBadSubtypeMatch:
534         stCap ~= FCAnullSt
535         AND (stEntry.openSeid ~= ofst.subtype
536             OR NOT mode SUBSET stEntry.openMode):
537         $(a subtype capability is specified, but it does not match the subtype
538         of the object, with access modes included)
539     KefcaDapViol: NOT SMXdap(pSeid, o, h_modeTrans(mode));
540     KefcaNotWritable: omWrite INSET mode AND isReadOnly(o);
541     KefcaNoExclDA:
542         omExclusive INSET mode AND NOT {omRead, omWrite} SUBSET mode;
543     KefcaCritExcl:
544         anotherExcl OR (openCount(o) > 0 AND omExclusive INSET mode):
545         $(either the object was opened elsewhere for exclusive use, or
546         exclusive use is requested and the object is open elsewhere)
547     KefcaOdSpace: nOpenDescriptors(pSeid) >= FCAMaxOpenDescriptors;
548 ASSERTIONS
549     FCAopenTableExists(pSeid);
550 EFFECTS
551     LET openDescriptor odl | FCAopenEntry(pSeid, odl) = ? AND odl ~= FCAnullSt
552     IN 'FCAopenEntry(pSeid, odl) = STRUCT(o, mode)
553     AND od = odl;
554
555 OFUN FCAclose(seid pSeid; openDescriptor od);                 $(FCAclose)
556     $(closes the open object named by "od". If the object is not in use by
557     anyone else, either by linking or by opening, then the object is deleted)
558 DEFINITIONS
559     openFileEntry oe IS FCAopenEntry(pSeid, od);
560     seid fSeid IS oe.openSeid;

```

```

561 globalData ofst IS FCAinfo(fSeid);
562 EXCEPTIONS
563 KEfcaBadOd: oe = ?;
564 ASSERTIONS
565 FCAopenTableExists(pSeid);
566 EFFECTS
567 ofst.linkCount = 0
568 AND openCount(fSeid) = 1
569 AND SENseidType(fSeid) = tFile
570 => 'FCAinfo(fSeid) = ?
571 AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?)
572 AND 'TIIinfo(fSeid) = ?;
573 'FCAopenEntry(pSeid, od) = ?;
574 $(the openAtCrash field is cleared when closing an object that was open
575 for writing. This has not been put in.)
576
577 $(----- open table maintenance -----)
578
579 OFUN FCAcreateOpenTable(seid pSeid); $(FCAcreateOpenTable)
580 $( this operation creates an open table associated with a process seid:
581 it is used as an auxiliary operation by the process modules when
582 creating a new a process)
583 ASSERTIONS
584 NOT FCAopenTableExists(pSeid);
585 EFFECTS
586 'FCAopenTableExists(pSeid) = TRUE;
587
588 OFUN FCAdeleteOpenTable(seid pSeid); $(FCAdeleteOpenTable)
589 $( Deletes the open table associated with a process; supports the release
590 of a process)
591 ASSERTIONS
592 FCAopenTableExists(pSeid):
593 EFFECTS
594 'FCAopenTableExists(pSeid) = FALSE;
595
596 $(----- utility operations -----)
597
598 OFUN FCAcloseAll(seid pSeid); $(FCAcloseAll)
599 $( Closes all the open objects of an open object table; supports process
600 release and invocation. May cause unreferenced objects to be deleted)
601 ASSERTIONS
602 FCAopenTableExists(pSeid):
603 EFFECTS
604 FORALL openDescriptor od | FCAopenEntry(pSeid, od) ~= ?;
605 seid fSeid = FCAopenEntry(pSeid, od).openSeid
606 'FCAopenEntry(pSeid, od) = ?
607 AND (openCount(fSeid) = 1
608 AND FCAinfo(fSeid).linkCount = 0
609 AND SENseidType(fSeid) = tFile)
610 => 'FCAinfo(fSeid) = ?
611 AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?)
612 AND 'TIIinfo(fSeid) = ?;
613 'FCAopenTableExists(pSeid) = FALSE;
614
615 OFUN FCAcopyOpenTable(seid fromSeid, toSeid); $(FCAcopyOpenTable)
616 $(Copies the contents of one open table, "fromSeid," to another, "toSeid.")

```

```

617     "toSeid" must be empty)
618     ASSERTIONS
619     FCAopenTableExists(fromSeid);
620     FCAopenTableExists(toSeid);
621     FORALL openDescriptor od : FCAopenEntry(toSeid, od) = ?;
622     EFFECTS
623     FORALL openDescriptor od
624         : 'FCAopenEntry(toSeid, od) = FCAopenEntry(fromSeid, od);
625
626     $(----- file status operations -----)
627
628     VFUN FCAGetFileStatus(seid pSeid, fSeid) -> fileStatus fst; $(FCAGetFileStatus)
629     $(returns the status of the file. The requesting process must have
630     mandatory access to the object, and the object must exist)
631     DEFINITIONS
632     seid f IS indir(fSeid);
633     globalData gl IS FCAinfo(f);
634     EXCEPTIONS
635     KEfcaNoFile: FCAinfo(f) = ? OR NOT SMXflow(pSeid, f, {daRead});
636     DERIVATION
637     STRUCT(IF FCAfileSize(f) = ? THEN 0 ELSE FCAfileSize(f),
638     gl.linkCount, gl.timeLastMod, gl.subtype,
639     gl.openAtCrash);
640
641     OFUN FCASetFileStatus(seid pSeid, fSeid; fileStatus newfs); $(FCASetFileStatus)
642     $(only the subtype file of an openable object may be changed. The
643     requesting process must have mandatory access to the object, and the
644     object must exist. Note the particular rule, explained below, relating
645     to subtypes)
646     DEFINITIONS
647     seid f IS indir(fSeid);
648     globalData oldfs IS FCAinfo(f);
649     EXCEPTIONS
650     KEfcaNoFile:
651     FCAinfo(f) = ? OR NOT SMXflow(pSeid, f, {daRead, daWrite});
652     KEfcaBadDa: NOT SMXdap(pSeid, f, {daRead, daWrite});
653     KEfcaNoOwner: TIIinfo(pSeid).owner ~= TIIinfo(f).owner;
654     KEfcaBadPriv: NOT SMXhasPriv(pSeid, privFileUpdateStatus);
655     KEfcaBadSubtype:
656     newfs.subtype ~= oldfs.subtype
657     AND (oldfs.subtype ~= nullStSeid
658     AND TIIinfo(oldfs.subtype).owner ~= TIIinfo(pSeid).owner
659     OR newfs.subtype ~= nullStSeid
660     AND TIIinfo(newfs.subtype).owner ~= TIIinfo(pSeid).owner);
661     $(the requesting process must be the owner of both the old and
662     new subtypes, if non-null)
663     KEfcaChangeLink: oldfs.linkCount ~= newfs.linkCount;
664     ASSERTIONS
665     FCAopenTableExists(pSeid);
666     EFFECTS
667     'FCAinfo(f) = STRUCT(newfs.linkCount, newfs.timeLastMod,
668     newfs.subtype, newfs.openAtCrash);
669
670
671     END MODULE

```

```

1  $( "      MODULE          fmi.specs (version 2.10)
2      CONTENTS:      File Miscellaneous Operations
3      TYPE:          SPECIAL specifications
4      LAST CHANGED:  10/12/79, 13:58:05
5  " )
6
7
8  MODULE fmi
9
10
11 $( this module contains miscellaneous file operations that are not included
12    in the fca module, because the SPECIAL checker at FACC cannot accomodate
13    the combined file)
14
15    TYPES
16
17    $(FROM smx)
18 nonDisType: STRUCT OF(
19             INTEGER securityLevel; SET_OF securityCat securityCatS;
20             INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
21 daType: SET_OF daMode;
22 modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
23 tiiStruct: STRUCT_OF(nonDisType nd: modeStruct da: INTEGER owner, group;
24                     SET_OF privType priv);
25
26    $(from fca)
27 globalData: STRUCT_OF(INTEGER linkCount, timeLastMod; seid subtype:
28                     BOOLEAN openAtCrash);
29 ioStatus: STRUCT_OF(INTEGER devIndep, devDep);
30 asyncId: CHAR;
31 fileBlock: VECTOR OF CHAR;
32 openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);
33 readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);
34 deviceStruct: STRUCT_OF(BOOLEAN addressable;
35                     INTEGER minRequest, maxRequest, modSize, maxBlockNo);
36 mountTableEntry: STRUCT_OF(seid leafSeid, rootSeid; BOOLEAN readOnly;
37                     tiiStruct devTii; globalData devGl);
38 fileSystemEntry: STRUCT_OF(seid fileSeid; globalData gl; tiiStruct tii;
39                     VECTOR_OF fileBlock fileData);
40 fileSystem: SET_OF fileSystemEntry;
41
42
43    DEFINITIONS
44
45 $(these definitions are explained in the fca module)
46
47 INTEGER FCAfileSize(seid fSeid) IS
48     CARDINALITY({INTEGER i | FCAfileData(fSeid, i) ~= ?});
49
50 INTEGER nOpenDescriptors(seid pSeid) IS
51     CARDINALITY({openDescriptor od | FCAopenEntry(pSeid, od) ~= ?});
52
53 INTEGER openCount(seid fSeid) IS
54     CARDINALITY({seid pSeid
55                 | (EXISTS openDescriptor od
56                   : FCAopenEntry(pSeid, od).openSeid = fSeid)});

```

```

57
58 deviceStruct deviceDataSeid(seid fSeid) IS
59   FCAdeviceData(FCAdeviceType(fSeid));
60
61 BOOLEAN isCurrentPath(seid tSeid) IS
62   EXISTS terminalGroup t : FCAcurrentPath(t) = tSeid;
63
64 seid indir(seid fSeid) IS
65   LET seid devSeid | FCAmountTable(devSeid).leafSeid = fSeid
66   IN IF devSeid = ? THEN fSeid ELSE FCAmountTable(devSeid).rootSeid;
67
68 BOOLEAN isReadOnly(seid fSeid) IS
69   EXISTS seid devSeid : SENseidNsp(FCAmountTable(devSeid).rootSeid)
70     = SENseidNsp(fSeid)
71     AND FCAmountTable(devSeid).readOnly = TRUE;
72
73
74   EXTERNALREFS
75
76   FROM smx:
77   seid: DESIGNATOR;
78   secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
79     tExtent, tNull};
80   privType: {
81     privFileUpdateStatus,   privLink,       privLockSeg,
82     privModifyPriv,         privMount,
83     privSetLevel,           privStickySeg,   privTerminalLock,
84     privViolSimpSecurity,   privViolStarSecurity,
85     privViolSimpIntegrity,  privViolStarIntegrity,
86     privViolDiscrAccess,    privSignal,    privWalkPTable,
87     privHalt,               privKernelCall, privViolCompartments,
88     privRealizeExecPermissions;
89   securityCat: DESIGNATOR;
90   integrityCat: DESIGNATOR;
91   daMode: {daRead, daWrite, daExecute};
92   VFUN SENseidNsp(seid s) -> INTEGER nsp;
93   VFUN SENseidType(seid s) -> secureEntityType set;
94   VFUN TIInfo(seid s) -> tiiStruct tii;
95   VFUN TIigetEntityLevel(seid pSeid, oSeid) -> tiiStruct otii;
96   OFUN TIisetEntityLevel(seid pSeid, oSeid; tiiStruct ntii);
97   VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
98   VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;
99   VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
100
101   FROM fca:
102   openDescriptor: DESIGNATOR;
103   openModes: {omRead, omWrite, omExclusive};
104   IOfunction: {rewind, etc};
105   deviceType: {RK05, RWPO4, RWPO5, RWPO6, RSWO4, TWE16, TM11, TU56, PR11,
106     PC11, LP11, IMP11B, LHDH};
107   terminalGroup: DESIGNATOR;
108   seid FCArootSeid;
109   VFUN FCAdeviceType(seid fSeid) -> deviceType d;
110   VFUN FCAdeviceData(deviceType d) -> deviceStruct ds;
111   VFUN FCAcurrentPath(terminalGroup t) -> seid s;
112   VFUN FCAterminalPathSet(terminalGroup t) -> SET_OF seid ss;

```

```

113 VFUN FCAmountTable(seid extentSeid) -> mountTableEntry mte:
114 VFUN FCAextentToFileSys(VECTOR_OF fileBlock fb; seid rootSeid)
115 -> fileSystem fs;
116 VFUN FCAinfo(seid fSeid) -> globalData gl:
117 VFUN FCAfileData(seid fSeid; INTEGER blockNo) -> fileBlock fb;
118 VFUN FCAinputStream(seid fSeid) -> VECTOR_OF CHAR vc;
119 VFUN FCAoutputStream(seid fSeid) -> VECTOR_OF CHAR vc;
120 VFUN FCAopenTableExists(seid pSeid) -> BOOLEAN b;
121 VFUN FCAopenEntry(seid pSeid; openDescriptor od) -> openFileEntry oe;
122
123
124 FUNCTIONS
125
126 $(----- process support operations -----)
127
128 $(Note: the function FCAcreateOpenTable is sufficient to support PROspawn,
129 and the function FCAcloseAll is sufficient to support PROinvoke.
130 The other support operations are listed here.)
131
132 OFUN FMIforkSupport(seid parent, child); $(FMIforkSupport)
133 $(This function creates a new open table, "child," and copies all open
134 from "parent" into it)
135 EXCEPTIONS
136 KEfmdExclusiveFile:
137 EXISTS openDescriptor od
138 : omExclusive INSET FCAopenEntry(parent, od).openMode;
139 ASSERTIONS
140 FCAopenTableExists(parent);
141 NOT FCAopenTableExists(child);
142 EFFECTS
143 'FCAopenTableExists(child) = TRUE;
144 FORALL openDescriptor od
145 : 'FCAopenEntry(child, od) = FCAopenEntry(parent, od);
146
147 OFUN FMireleaseSupport(seid pSeid); $(FCAreleaseSupport)
148 $(Closes all the open objects of an open object table and deletes the
149 table)
150 ASSERTIONS
151 FCAopenTableExists(pSeid);
152 EFFECTS
153 FORALL openDescriptor od | FCAopenEntry(pSeid, od) ~= ?;
154 seid fSeid = FCAopenEntry(pSeid, od).openSeid
155 : 'FCAopenEntry(pSeid, od) = ?
156 AND (openCount(fSeid) = 1
157 AND FCAinfo(fSeid).linkCount = 0
158 AND SENseidType(fSeid) = tFile)
159 => 'FCAinfo(fSeid) = ?
160 AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?)
161 AND 'TIIinfo(fSeid) = ?;
162 'FCAopenTableExists(pSeid) = FALSE;
163
164 $(----- link and unlink operations -----)
165
166 OFUN FCAlink(seid pSeid, fSeid); $(FCAlink)
167 $(increments the link count of an existing file. Mandatory access is
168 required, and the process must have the privilege to link. The file

```

```

169     system must not be mounted in read only mode)
170 DEFINITIONS
171     seid f IS indir(fSeid);
172     globalData fs IS FCAinfo(f);
173 EXCEPTIONS
174     KEfcaBadPriv: NOT SMXhasPriv(pSeid, privLink);
175     KEfcaNotThere: fs = ? OR NOT SMXflow(pSeid, f, {daRead, daWrite});
176     KEfcaNotFile: SENseidType(f) ~= tFile;
177     KEfcaReadOnly: isReadOnly(f);
178 ASSERTIONS
179     FCAopenTableExists(pSeid);
180 EFFECTS
181     'FCAinfo(f)
182     = STRUCT(fs.linkCount + 1, fs.timeLastMod, fs.subtype, fs.openAtCrash);
183
184 OFUN FCAunlink(seid pSeid, fSeid);                                $(FCAunlink)
185 $(decrements the reference count of an existing file. Mandatory access is
186 required and the process must have the privilege to link. The file
187 system that the file is on must not be mounted in read only mode. Note
188 that unlinking can cause the file to be deleted if the link count goes
189 to zero and the file is not open anywhere)
190 DEFINITIONS
191     seid f IS indir(fSeid);
192     globalData fs IS FCAinfo(f);
193 EXCEPTIONS
194     KEfcaBadpriv: NOT SMXhasPriv(pSeid, privLink);
195     KEfcaNotThere: fs = ? OR NOT SMXflow(pSeid, f, {daRead, daWrite});
196     KEfcaNotFile: SENseidType(f) ~= tFile;
197     KEfcaReadOnly: isReadOnly(f);
198 ASSERTIONS
199     FCAopenTableExists(pSeid);
200 EFFECTS
201     IF fs.linkCount = 1 AND openCount(f) = 0
202     THEN 'FCAinfo(f) = ?
203         AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?)
204         AND 'TIIinfo(fSeid) = ?
205     ELSE 'FCAinfo(f)
206         = STRUCT(fs.linkCount - 1, fs.timeLastMod, fs.subtype,
207                 fs.openAtCrash);
208
209 $(----- basic I/O operations -----)
210
211 VFUN FCAvReadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size:
212                     asyncId id)                                $(FCAvReadBlocks)
213     -> readResult rr;
214 $(the purpose of this function is to return the result that FCAreadBlocks
215 would return if executed)
216 $(returns blocks that are read from a given file, device, extent, or
217 terminal. the object must be open for reading. the block number and
218 size must be within range for the kind of object specified. note
219 that files, extents, and addressable devices are handled differently
220 from terminals and nonaddressable devices, with respect to the data.)
221 DEFINITIONS
222     seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
223     deviceStruct d
224     IS IF SENseidType(fSeid) = tDevice THEN deviceDataSeid(fSeid)

```

```

225     ELSE IF SENSeidType(fSeid) INSET {tFile, tExtent}
226     THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid)-1)
227     ELSE STRUCT(FALSE, 1, 255, 1, 0); $(tTerminal)
228 EXCEPTIONS
229 KEfcaBadOd: FCAopenEntry(pSeid, od) = ?;
230 KEfca adType: SENSeidType(fSeid) = tSubtype;
231 KEfcaNotReadable: NOT omRead INSET FCAopenEntry(pSeid, od).openMode;
232 KEfcaTermLocked:
233     SENSeidType(fSeid) = tTerminal AND NOT isCurrentPath(fSeid);
234 KEfcaBadSize:
235     NOT size INSET {d.minRequest .. d.maxRequest}
236     OR (size MOD d.modSize) ~= 0;
237 KEfcaBadBlockNo: NOT blockNo INSET {0 .. d.maxBlockNo};
238 KEfcaEndOfFile:
239     d.addressable AND blockNo + size/d.modSize > d.maxBlockNo;
240 ASSERTIONS
241 FCAopenTableExists(pSeid);
242 DERIVATION
243 LET INTEGER sl INSET {0 .. size/d.modSize}
244 IN STRUCT(IF d.addressable
245     THEN VECTOR(FOR i FROM blockNo TO blockNo + sl - 1
246         : FCAfileData(fSeid, i))
247     ELSE VECTOR(FOR i FROM 1 TO sl $(d.modSize = 1)
248         : VECTOR(FCAinputStream(fSeid)[i]))
249     SOME ioStatus ios | TRUE);
250
251 OVFUN FCAreadBlocks(seid pSeid: openDescriptor od; INTEGER blockNo, size;
252     asyncId id)
253     -> readResult rr; $(FCAreadBlocks)
254 $(LR -- needs semantics for asynchronous I/O)
255 $(returns blocks that are read from a given file, device, extent, or
256     terminal. the object must be open for reading. the block number and
257     size must be within range for the kind of object specified. note
258     that files, extents, and addressable devices are handled differently
259     from terminals and nonaddressable devices, with respect to the data.)
260 DEFINITIONS
261 seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
262 deviceStruct d
263     IS IF SENSeidType(fSeid) = tDevice THEN deviceDataSeid(fSeid)
264     ELSE IF SENSeidType(fSeid) INSET {tFile, tExtent}
265     THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid)-1)
266     ELSE STRUCT(FALSE, 1, 255, 1, 0); $(tTerminal)
267 readResult rrl IS FCAvReadBlocks(pSeid, od, blockNo, size, id);
268 EXCEPTIONS
269 EXCEPTIONS_OF FCAvReadBlocks(pSeid, od, blockNo, size, id);
270 ASSERTIONS
271 FCAopenTableExists(pSeid);
272 EFFECTS
273 rr = rrl;
274 NOT d.addressable
275 => 'FCAinputStream(fSeid)
276     = VECTOR(FOR i FROM LENGTH(rrl.data) + 1
277         TO LENGTH(FCAinputStream(fSeid))
278         : FCAinputStream(fSeid)[i]);
279
280 OVFUN FCAwriteBlocks(seid pSeid: openDescriptor od; INTEGER blockNo:

```

```

281          VECTOR OF fileBlock vfb; asyncId id)
282  -> ioStatus ios:                                     $(FCAwriteBlocks)
283  $(LR -- needs asynchronous I/O)
284  $(writes the contents of a vector of file blocks onto the object mentioned.
285  the data must correspond to the parameters of the openable object.)
286  DEFINITIONS
287    seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
288    globalData fs IS FCAinfo(fSeid);
289    secureEntityType type IS SENseidType(fSeid);
290    deviceStruct d
291      IS IF type = tDevice THEN deviceDataSeid(fSeid)
292      ELSE IF type INSET {tFile, tExtent}
293      THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid))
294      ELSE STRUCT(FALSE, 1, 255, 1, 0);
295  EXCEPTIONS
296    KEfcaBadOd: FCAopenEntry(pSeid, od) = ?;
297    KEfcaNotWritable: NOT onWrite INSET FCAopenEntry(pSeid, od).openMode;
298    KEfcaTermLocked: type = tTerminal AND NOT isCurrentPath(fSeid);
299    KEfcaBadRequest:
300      NOT LENGTH(vfb) INSET {d.minRequest .. d.maxRequest}
301      OR (LENGTH(vfb) MOD d.modSize) ~= 0;
302    KEfcaBadBlockNo: NOT blockNo INSET {0 .. d.maxBlockNo};
303    KEfcaOverflow: d.addressable AND type ~= tFile
304      AND blockNo + LENGTH(vfb) > d.maxBlockNo;
305  EFFECTS
306    ios = (SOME ioStatus ios1 | TRUE);
307    EXISTS INTEGER sizel INSET {0 .. LENGTH(vfb)}
308      : IF d.addressable
309      THEN FORALL INTEGER i
310        : 'FCAfileData(fSeid, i)
311          = (IF NOT i INSET {blockNo .. blockNo + sizel - 1}
312            THEN FCAfileData(fSeid, i)
313            ELSE vfb[i - blockNo + 1])
314        ELSE 'FCAoutputStream(fSeid)
315          = VECTOR(FOR i FROM 1
316                    TO LENGTH(FCAoutputStream(fSeid))+LENGTH(vfb)
317                    IF i <= LENGTH(vfb)
318                    THEN vfb[i][1]
319                    ELSE FCAoutputStream(fSeid)[i+LENGTH(vfb)]);
320
321  $(----- general device manipulation -----)
322
323  VFUN FCAvDeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
324    VECTOR OF INTEGER args; asyncId id)
325    -> ioStatus status;                                     $(FCAvDeviceFunction)
326    $(the purpose of this function is to return the value that FCAdeviceFunction
327    would return if it were executed in a given state)
328    $(the specification of this function is device-dependent, but may be
329    filled in at a later time)
330  DEFINITIONS
331    seid dSeid IS FCAopenEntry(pSeid, od).openSeid;
332  EXCEPTIONS
333    KEfcaNotThere: FCAopenEntry(pSeid, od) = ?;
334    KEfcaBadDevice: NOT SENseidType(dSeid) INSET {tDevice, tTerminal};
335    KEfcaNoOwner: TIIinfo(dSeid).owner ~= TIIinfo(pSeid).owner;
336  ASSERTIONS

```

```

337   FCAopenTableExists(pSeid):
338   DERIVATION
339   SOME ioStatus ios | TRUE;
340
341   OVFUN FCADeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
342   VECTOR_OF INTEGER args; asyncId id)
343   -> ioStatus status: $(FCADeviceFunction)
344   $(the specification of this function is device-dependent, but may be
345   filled in at a later time)
346   EXCEPTIONS
347   EXCEPTIONS_OF FCAvDeviceFunction(pSeid, od, f, args, id);
348   ASSERTIONS
349   FCAopenTableExists(pSeid);
350   EFFECTS
351   $(the state of the device somehow changes, and the result is returned via
352   the io status)
353   status = FCAvDeviceFunction(pSeid, od, f, args, id);
354   TRUE;
355
356   OFUN FCATerminalLock(seid pSeid, devSeid); $(FCATerminalLock)
357   $(sets the current terminal in the group to be "devSeid". The requesting
358   process must have the privilege to lock terminals)
359   EXCEPTIONS
360   KEfcaNotTerminal: SENSeidType(devSeid) ~= tTerminal;
361   KEfcaNoPriv: NOT SMXhasPriv(pSeid, privTerminalLock);
362   KEfcaNotThere: FCAinfo(devSeid) = ?;
363   ASSERTIONS
364   FCAopenTableExists(pSeid);
365   EFFECTS
366   LET terminalGroup t | devSeid INSET FCATerminalPathSet(t)
367   IN 'FCAcurrentPath(t) = devSeid;
368
369   $(----- mounting and unmounting operations -----)
370
371   OFUN FCAMount(seid pSeid, dev, leaf, root; BOOLEAN readOnly); $(FCAMount)
372   $(performs the logical mounting of a file system from an extent. The
373   semantics are described in the general commentary in the FCA
374   specification)
375   DEFINITIONS
376   fileSystem fileSys IS
377   FCAextentToFileSys(VECTOR(FOR i FROM 1 TO FCAfileSize(dev)
378   : FCAfileData(dev, i)),
379   root);
380   $(the file system produced by the data on the extent)
381   fileSystemEntry fse(seid f) IS
382   SOME fileSystemEntry fsel | fsel.fileSeid = f:
383   $(the entry in the file system with a given seid)
384   EXCEPTIONS
385   KEfcaBadPriv: NOT SMXhasPriv(pSeid, privMount);
386   KEfcaNoLeaf: FCAinfo(leaf) = ?
387   OR NOT SMXflow(pSeid, leaf, {daRead, daWrite});
388   KEfcaNoDev: FCAinfo(dev) = ?
389   OR NOT SMXflow(pSeid, dev, {daRead, daWrite});
390   KEfcaBadDal: NOT SMXdap(pSeid, dev, {daWrite});
391   KEfcaBadDa2: NOT SMXdap(pSeid, leaf, {daWrite});
392   KEfcaNoFile1: SENSeidType(leaf) ~= tFile;

```

```

393 KEfcaNoFile2: SENseidType(root) ~= tFile;
394 KEfcaNoExtent: SENseidType(leaf) ~= tExtent;
395 KEfcaInUse:
396     SENseidNsp(root) = SENseidNsp(FCARootSeid)
397     OR (EXISTS seid devSeid
398         : SENseidNsp(FCAmountTable(devSeid).rootSeid)
399         = SENseidNsp(root));
400 KEfcaNoOwner: TIIinfo(pSeid).owner ~= TIIinfo(dev).owner;
401 KEfcaFileOpen: openCount(dev) > 0;
402 KEfcaDevOpen: openCount(leaf) > 0;
403 RESOURCE_ERROR;
404 ASSERTIONS
405     FCAopenTableExists(pSeid);
406 EFFECTS
407     'FCAmountTable(dev)
408     = STRUCT(leaf, root, readOnly, TIIinfo(dev), FCAinfo(dev));
409     'FCAinfo(dev) = ?;
410     FORALL INTEGER i : 'FCAfileData(dev, i) = ?;
411     'TIIinfo(dev) = ?;
412     FORALL seid f | fse(f) ~= ?
413     : 'FCAinfo(f) = fse(f).gl
414       AND (FORALL INTEGER i INSET {1 .. LENGTH(fse(f).fileData)}
415           : 'FCAfileData(f, i - 1) = fse(f).fileData[i])
416       AND 'TIIinfo(f) = fse(f).tii;
417
418 OFUN FCAunmount(seid pSeid, devSeid);                                $(FCAunmount)
419 $(performs logical unmounting of a file system. Restores the extent
420  so that it can be accessed)
421 DEFINITIONS
422     mountTableEntry mte IS FCAmountTable(devSeid);
423     INTEGER fsNsp IS SENseidNsp(mte.rootSeid);
424     fileSystem fs IS
425     {fileSystemEntry fse
426       | LET seid fSeid | SENseidNsp(fSeid) = fsNsp AND FCAinfo(fSeid) ~= ?
427       IN fse = STRUCT(fSeid, FCAinfo(fSeid), TIIinfo(fSeid),
428                       VECTOR(FOR i FROM 1 TO FCAfileSize(fSeid)
429                             : FCAfileData(fSeid, i - 1))));
430     $( the state of the file system to be unmounted)
431     VECTOR OF fileBlock extData
432     IS SOME VECTOR OF fileBlock vfb
433       | FCAextentToFileSys(vfb,
434                             SOME seid s
435                             | EXISTS fileSystemEntry fse INSET fs
436                               : s = fse.fileSeid)
437       = fs;
438     $(given the state of the file system, the extent that is equivalent to
439     it)
440 EXCEPTIONS
441     KEfcaBadPriv: NOT SMXhasPriv(pSeid, privMount);
442     KEfcaNoDevice: mte = ?
443     OR NOT SMXflow(pSeid, devSeid, {daRead, daWrite});
444     KEfcaNoOwner: TIIinfo(pSeid).owner = TIIinfo(devSeid).owner;
445     KEfcaBadDa: NOT SMXdap(pSeid, devSeid, {daWrite});
446     KEfcaOpenFiles:
447     EXISTS seid fSeid | SENseidNsp(fSeid) = fsNsp : openCount(fSeid) > 0;
448 ASSERTIONS

```

```
449 FCAopenTableExists(pSeid):
450 EFFECTS
451   'FCAinfo(devSeid) = mte.devGl;
452   'TIIinfo(devSeid) = mte.devTii;
453   FORALL INTEGER i INSET {1 .. LENGTH(extData)}
454     : 'FCAfileData(devSeid, i - 1) = extData[i];
455   $(the device's state "comes back")
456   FORALL seid fSeid | SENseidNsp(fSeid) = fsNsp
457     : 'TIIinfo(fSeid) = ?
458     AND 'FCAinfo(fSeid) = ?
459     AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?);
460   $(the state of all files on the file system "disappears")
461
462
463 END MODULE
```

```

1 S("      MODULE      ker.specs (version 2.21)
2      CONTENTS:      Kernel Calls
3      TYPE      SPECIAL.specifications
4      LAST CHANGED  10/17/79, 18:56:31
5      ")
6
7
8 MODULE ker
9
10
11     TYPES
12
13     $(from mac)
14 vAddrType: {0 .. MACmaxVAddr};
15
16     $(from smx)
17 nonDisType: STRUCT OF(
18         INTEGER securityLevel; SET OF securityCat securityCatS;
19         INTEGER integrityLevel; SET OF integrityCat integrityCatS);
20 daType: SET OF daMode;
21 modeStruct: STRUCT OF(daType ownerMode, groupMode, allMode);
22 ti1Struct: STRUCT OF(nonDisType nd;
23         modeStruct da; INTEGER owner, group: SET OF privType priv);
24
25     $(FROM pvm)
26 virtualLocation:
27     STRUCT OF(domainType domain; spaceType idSpace; vAddrType vAddr);
28 globalData:
29     STRUCT OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
30         direction growth);
31 segStatus: STRUCT OF(globalData gl; INTEGER size);
32 pBlock: STRUCT OF(virtualLocation vloc; INTEGER size);
33
34     $(FROM fca)
35 fileStatus: STRUCT OF(INTEGER nBlocks, linkCount, timeLastMod; seid subtype;
36         BOOLEAN openForWrite, openAtCrash);
37 asyncId: CHAR;
38 ioStatus: STRUCT OF(INTEGER devIndep, devDep);
39
40     $(from pro)
41 piLevelType: {0 .. PROMaxPiLevel}; $(pseudo interrupt level range)
42 piEntryType: STRUCT OF(BOOLEAN pending;
43         piLevelType oldPIL;
44         INTEGER oldPc;
45         INTEGER oldPs;
46         INTEGER parameter;
47         INTEGER newPc;
48         INTEGER newPs);
49 piVectorType: {VECTOR OF piEntryType piv | LENGTH(piv) = PROMaxPiLevel + 1};
50 ipcqType: {VECTOR OF ipcMessageType zz | LENGTH(zz) <= IPCmaxMessageCount};
51 ipcTextType: {VECTOR OF CHAR vc | LENGTH(vc) = IPCmaxMessageLength};
52 ipcMessageType: STRUCT OF(seid sender; ipcTextType text);
53 processStateType: STRUCT OF(seid self;
54         seid parent;
55         INTEGER family;
56         INTEGER realUser;

```

```

57             INTEGER realGroup;
58             INTEGER pc;           $(program counter)
59             INTEGER ps;           $(processor status)
60             piLevelType pil;
61             piVectorType piv;
62             ipcqType ipcq;
63             INTEGER advPrio;       $(advisory priority)
64             INTEGER timerAlarm;    $(one-zero crossing => pi)
65             INTEGER supervisorTiming;
66             INTEGER userTiming;
67             BOOLEAN timTog;        $(timer toggle TRUE is ON)
68
69
70     EXTERNALREFS
71
72     FROM mac:
73     INTEGER MACmaxVAddr;
74
75     FROM smx:
76     seid: DESIGNATOR;
77     privType {
78         privFileUpdateStatus,   privLink,           privLockSeg,
79         privModifyPriv,         privMount,
80         privSetLevel,           privStickySeg,       privTerminalLock,
81         privViolSimpSecurity,   privViolStarSecurity,
82         privViolSimpIntegrity,  privViolStarIntegrity,
83         privViolDiscrAccess,    privSignal,        privWalkPTable,
84         privHalt,               privKernelCall,   privViolCompartments,
85         privRealizeExecPermissions;
86     daMode: {daRead, daWrite, daExecute};
87     securityCat: DESIGNATOR;
88     integrityCat: DESIGNATOR;
89     domainType: {userDomain, supervisorDomain};
90
91     FROM pvm:
92     segDes: DESIGNATOR;
93     spaceType {iSpace, dSpace};
94     direction: {up, down};
95     OVFUN PVMbuild(seid pSeid; segStatus ss; modeStruct ms; INTEGER size;
96                 virtualLocation vl)
97     -> STRUCT OF(seid segSeid; segDes segd) result;
98     OFUN PVMdestroy(seid pSeid; segDes segd);
99     VFUN PVMgetSegmentStatus(seid pSeid, segSeid) -> segStatus st;
100    OFUN PVMsetSegmentStatus(seid pSeid, segSeid; segStatus st);
101    OFUN PVMremap(seid pSeid; segDes in; virtualLocation vl; daType da;
102                BOOLEAN vlFlg, daFlg; segDes out; INTEGER outSize:
103                BOOLEAN osFlg);
104    OVFUN PVMrendezvous(seid pSeid, segSeid; virtualLocation vl; daType da)
105    -> segDes segd;
106
107    FROM fca:
108    openDescriptor: DESIGNATOR;
109    openModes: {omRead, omWrite, omExclusive};
110    IOfunction: {rewind, etc};
111    OFUN FCAClose(seid pSeid; openDescriptor od);
112    OVFUN FCAcreate(seid pSeid, nspSeid; modeStruct ms; openDescriptor stCap)

```

```

113  -> STRUCT OF(seid fSeid: openDescriptor od) result:
114  OVFUN FCAopen(seid pSeid, oSeid; SET OF openModes om: openDescriptor stCap)
115  -> openDescriptor od;
116  VFUN FCAgetFileStatus(seid pSeid, fSeid) -> fileStatus fst;
117  OFUN FCAsetFileStatus(seid pSeid, fSeid: fileStatus newfst);
118
119  FROM fmi:
120  OFUN FCAlink(seid pSeid, fSeid):
121  OFUN FCAmount(seid pSeid, dev, leaf, root; BOOLEAN readOnly):
122  OFUN FCAterminalLock(seid pSeid, devSeid);
123  OFUN FCAunlink(seid pSeid, fSeid);
124  OFUN FCAunmount(seid pSeid, devSeid);
125
126
127  FROM pro:
128  INTEGER IPCmaxMessageCount;
129  INTEGER IPCmaxMessageLength;
130  INTEGER PROMaxPILevel;
131  OVFUN PROfork(seid pSeid) -> seid childSeid;
132  VFUN PROgetProcessStatus(seid pSeid, getSeid) -> processStateType ps;
133  OFUN PROinterruptReturn(seid pSeid);
134  OFUN PROinvoke(seid pSeid, immSeid; segDes arg);
135  OFUN PROnap(seid pSeid; INTEGER timeOut);
136  OFUN PROpost(seid pSeid, receiver; BOOLEAN pseudoInt: ipcTextType msg);
137  OVFUN PROreceive(seid pSeid; INTEGER timeOut) -> ipcMessageType msg;
138  OFUN PROreleaseProc(seid pSeid, rSeid);
139  OFUN PROsetProcessStatus(seid pSeid, procSeid; processStateType ps);
140  OFUN PROsignal(seid pSeid, procSeid; INTEGER sigVal);
141  OVFUN PROspawn(seid pSeid, immSeid; segDes arg) -> seid childSeid;
142  VFUN PROwalkProcessTable(seid pSeid; INTEGER n) -> seid rSeid;
143
144  FROM lev:
145  VFUN LEVgetObjectLevel(seid pSeid, objSeid) -> ttiStruct level;
146  OFUN LEVsetObjectLevel(seid pSeid, objSeid; ttiStruct level);
147
148  FROM spf:
149  SPFFunctionType: {syncSPF, immSegSPF, sysHaltSPF, levelSetSPF};
150
151  FROM pbl:
152  OFUN PBLdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
153      pBlock arguments, status; asyncId id);
154  OVFUN PBLreadBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
155      pBlock duFile; asyncId as)
156  -> STRUCT OF(INTEGER bytesRead; ioStatus errst) result:
157  OFUN PBLspecialFunction(seid pSeid; SPFFunctionType fn; pBlock parm);
158  OFUN PBLwriteBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
159      pBlock duFile; asyncId as);
160
161
162  FUNCTIONS
163
164  $(Visible Kernel Functions in Alphabetical Order)
165
166  OVFUN K build_segment(segStatus ss; modeStruct ms; INTEGER size;
167      virtualLocation vl)
168      [seid pSeid] -> STRUCT OF(seid segSeid; segDes segd) result:

```

```

169                                     $(K_build_segment)
170 EXCEPTIONS
171 EXCEPTIONS OF PVMbuild(pSeid, ss, ms, size, vl);
172 EFFECTS
173 result = EFFECTS OF PVMbuild(pSeid, ss, ms, size, vl);
174
175 OFUN K_close(openDescriptor od)[seid pSeid]; $(K_close)
176 EXCEPTIONS
177 EXCEPTIONS OF FCAClose(pSeid, od);
178 EFFECTS
179 EFFECTS OF FCAClose(pSeid, od);
180
181 OVFUN K_create(seid nspSeid; modeStruct ms; openDescriptor stCap)[seid pSeid]
182 -> STRUCT OF(seid fSeid; openDescriptor od) return; $(K_create)
183 EXCEPTIONS
184 EXCEPTIONS OF FCACreate(pSeid, nspSeid, ms, stCap);
185 EFFECTS
186 return = EFFECTS OF FCACreate(pSeid, nspSeid, ms, stCap);
187
188 OFUN K_device_function(openDescriptor od; IOfunction f;
189 pBlock arguments, status; asyncId id)
190 [seid pSeid]; $(K_device_function)
191 EXCEPTIONS
192 EXCEPTIONS OF PBLdeviceFunction(pSeid, od, f, arguments, status, id);
193 EFFECTS
194 EFFECTS OF PBLdeviceFunction(pSeid, od, f, arguments, status, id);
195
196 OVFUN K_fork()[seid pSeid] -> seid childSeid; $(K_fork)
197 EXCEPTIONS
198 EXCEPTIONS OF PROfork(pSeid);
199 EFFECTS
200 childSeid = EFFECTS OF PROfork(pSeid);
201 $("ChildSeid is returned to the (original) process: pSeid (parent seid)
202 is returned to the newly created child")
203
204 VFUN K_get_file_status(seid fSeid)[seid pSeid] -> fileStatus fst;
205                                     $(K_get_file_status)
206 EXCEPTIONS
207 EXCEPTIONS OF FCAGetFileStatus(pSeid, fSeid);
208 DERIVATION
209 FCAGetFileStatus(pSeid, fSeid);
210
211 VFUN K_get_object_level(seid obSeid)[seid pSeid] -> ttiStruct otii;
212                                     $(K_get_object_level)
213 EXCEPTIONS
214 EXCEPTIONS OF LEVgetObjectLevel(pSeid, obSeid);
215 DERIVATION
216 LEVgetObjectLevel(pSeid, obSeid);
217
218 VFUN K_get_process_status(seid getSeid)[seid pSeid] -> processStateType ps;
219                                     $(K_get_process_status)
220 EXCEPTIONS
221 EXCEPTIONS OF PROgetProcessStatus(pSeid, getSeid);
222 DERIVATION
223 PROgetProcessStatus(pSeid, getSeid);
224

```

```

225 VFUN K_get_segment_status(seid segSeid)[seid pSeid] -> segStatus st:
226                                     $(K_get_segment_status)
227 EXCEPTIONS
228     EXCEPTIONS_OF PVMgetSegmentStatus(pSeid, segSeid);
229 DERIVATION
230     PVMgetSegmentStatus(pSeid, segSeid);
231
232 OFUN K_interrupt_return()[seid pSeid];                                     $(K_interrupt_return)
233 EXCEPTIONS
234     EXCEPTIONS_OF PROinterruptReturn(pSeid);
235 EFFECTS
236     EFFECTS_OF PROinterruptReturn(pSeid);
237
238 OFUN K_invoke(seid immSeid; segDes arg)[seid pSeid];                       $(K_invoke)
239 EXCEPTIONS
240     EXCEPTIONS_OF PROinvoke(pSeid, immSeid, arg);
241 EFFECTS
242     EFFECTS_OF PROinvoke(pSeid, immSeid, arg);
243
244 OFUN K_link(seid fSeid)[seid pSeid];                                       $(K_link)
245 EXCEPTIONS
246     EXCEPTIONS_OF FCAlink(pSeid, fSeid);
247 EFFECTS
248     EFFECTS_OF FCAlink(pSeid, fSeid);
249
250 OFUN K_mount(seid dev, leaf, root; BOOLEAN readOnly)[seid pSeid]: $(K_mount)
251 EXCEPTIONS
252     EXCEPTIONS_OF FCAmount(pSeid, dev, leaf, root, readOnly);
253 EFFECTS
254     EFFECTS_OF FCAmount(pSeid, dev, leaf, root, readOnly);
255
256 OFUN K_nap(INTEGER timeOut)[seid pSeid];                                   $(K_nap)
257 EXCEPTIONS
258     EXCEPTIONS_OF PROnap(pSeid, timeOut);
259 EFFECTS
260     EFFECTS_OF PROnap(pSeid, timeOut);
261
262 OVFUN K_open(seid oSeid; SET_OF openModes om; openDescriptor stCap)
263     [seid pSeid] -> openDescriptor od;                                     $(K_open)
264 EXCEPTIONS
265     EXCEPTIONS_OF FCAopen(pSeid, oSeid, om, stCap);
266 EFFECTS
267     od = EFFECTS_OF FCAopen(pSeid, oSeid, om, stCap);
268
269 OFUN K_post(seid receiver; BOOLEAN pseudoInt; ipcTextType msg)[seid pSeid];
270                                                         $(K_post)
271 EXCEPTIONS
272     EXCEPTIONS_OF PROpost(pSeid, receiver, pseudoInt, msg);
273 EFFECTS
274     EFFECTS_OF PROpost(pSeid, receiver, pseudoInt, msg);
275
276 OVFUN K_read_block(openDescriptor od; INTEGER blockNo; pBlock duFile;
277     asyncId as)[seid pSeid]
278 -> STRUCT_OF(INTEGER bytesRead; ioStatus errst) result;    $(K_read_block)
279 EXCEPTIONS
280     EXCEPTIONS_OF PBLreadBlock(pSeid, od, blockNo, duFile, as);

```

www.elsevier.com/locate/jmb

Abstract

100

1

—

1

10

—

Keywords:

10

```

337 EXCEPTIONS
338 EXCEPTIONS OF LEVsetObjectLevel(pSeid, objSeid, level):
339 EFFECTS
340 EFFECTS OF LEVsetObjectLevel(pSeid, objSeid, level):
341
342 OFUN K_set_process_status(seid procSeid; processStateType ps)[seid pSeid];
343                                     $(K_set_process_status)
344 EXCEPTIONS
345 EXCEPTIONS OF PROsetProcessStatus(pSeid, procSeid, ps):
346 EFFECTS
347 EFFECTS OF PROsetProcessStatus(pSeid, procSeid, ps):
348
349 OFUN K_set_segment_status(seid segSeid; segStatus st)[seid pSeid];
350                                     $(K_set_segment_status)
351 EXCEPTIONS
352 EXCEPTIONS OF PVMsetSegmentStatus(pSeid, segSeid, st):
353 EFFECTS
354 EFFECTS OF PVMsetSegmentStatus(pSeid, segSeid, st):
355
356 OFUN K_signal(seid procSeid; INTEGER sigVal)[seid pSeid];           $(K_signal)
357 EXCEPTIONS
358 EXCEPTIONS OF PROsignal(pSeid, procSeid, sigVal):
359 EFFECTS
360 EFFECTS OF PROsignal(pSeid, procSeid, sigVal):
361
362 OVFUN K_spawn(seid immSeid; segDes arg)[seid pSeid] -> seid childSeid;
363                                     $(K_spawn)
364 EXCEPTIONS
365 EXCEPTIONS OF PROspawn(pSeid, immSeid, arg):
366 EFFECTS
367 childSeid = EFFECTS_OF PROspawn(pSeid, immSeid, arg);
368
369 OFUN K_special_function(SPFFunctionType fn; pBlock parm)[seid pSeid];
370                                     $(K_special_function)
371 EXCEPTIONS
372 EXCEPTIONS OF PBLspecialFunction(pSeid, fn, parm):
373 EFFECTS
374 EFFECTS OF PBLspecialFunction(pSeid, fn, parm):
375
376 OFUN K_unlink(seid fSeid)[seid pSeid];                               $(K_unlink)
377 EXCEPTIONS
378 EXCEPTIONS OF FCAunlink(pSeid, fSeid):
379 EFFECTS
380 EFFECTS OF FCAunlink(pSeid, fSeid):
381
382 OFUN K_unmount(seid devSeid)[seid pSeid];                           $(K_unmount)
383 EXCEPTIONS
384 EXCEPTIONS OF FCAunmount(pSeid, devSeid):
385 EFFECTS
386 EFFECTS OF FCAunmount(pSeid, devSeid):
387
388 VFUN K_walk_process_table(INTEGER n)[seid pSeid] -> seid rSeid;
389                                     $(K_walk_process_table)
390 EXCEPTIONS
391 EXCEPTIONS OF PROwalkProcessTable(pSeid, n):
392 DERIVATION

```

```
393     PROwalkProcessTable(pSeid,n);
394
395 OFUN K_write_block(openDescriptor od; INTEGER blockNo; pBlock duFile;
396                     asyncId id)[seid pSeid];           $(K write block)
397 EXCEPTIONS
398     EXCEPTIONS OF PBLwriteBlock(pSeid, od, blockNo, duFile, id);
399 EFFECTS
400     EFFECTS OF PBLwriteBlock(pSeid, od, blockNo, duFile, id);
401
402
403 END MODULE
```

```

1  $( "      MODULE:      lev.specs (version 2.3)
2           CONTENTS:    System Levels
3           TYPE:        SPECIAL specifications
4           LAST CHANGED: 7/17/79, 15:04:04
5  " )
6
7
8  MODULE lev
9
10
11 $( This module enables the centralization of the get and set level operations
12    for all modules. Each module maintains the type-independent operation
13    of its own objects, and applies certain conditions to the getting and
14    setting of this information. However, there is only one kernel operation
15    for getting levels, and one for setting levels, of all objects. The
16    operations are combined in this module)
17
18    TYPES
19
20    $(from smx)
21    nonDisType: STRUCT_OF(
22                INTEGER securityLevel; SET_OF securityCat securityCatS:
23                INTEGER integrityLevel. SET_OF integrityCat integrityCatS);
24    daType: SET_OF daMode;
25    modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
26    ttiStruct: STRUCT_OF(nonDisType nd; modeStruct da; INTEGER owner, group;
27                        SET_OF privType priv);
28
29    $(FROM pvm)
30    globalData:
31    STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
32              direction growth);
33
34    $(FROM fca)
35    openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);
36
37
38    EXTERNALREFS
39
40    FROM smx:
41    seid: DESIGNATOR;
42    secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
43                      tExtent, tNull};
44    securityCat: DESIGNATOR;
45    integrityCat: DESIGNATOR;
46    daMode: {daRead, daWrite, daExecute};
47    privType: {
48                privFileUpdateStatus,    privLink,        privLockSeg,
49                privModifyPriv,           privMount,
50                privSetLevel,             privStickySeg,    privTerminalLock,
51                privViolSimpSecurity,     privViolStarSecurity,
52                privViolSimpIntegrity,    privViolStarIntegrity,
53                privVio!DiscrAccess,     privSignal,      privWalkPTable,
54                privHalt,                 privKernelCall,  privViolCompartments,
55                privRealizeExecPermissions};
56    VFUN SENSEidType(seid s) -> secureEntityType set;

```

```
57 VFUN TIIgetEntityLevel(seid pSeid, oSeid) -> tiiStruct ntii:
58 OFUN TIIsetEntityLevel(seid pSeid, oSeid; tiiStruct ntii):
59
60 FROM pvm:
61 direction: {up, down}:
62 VFUN SEGinstanceInfo(seid s) -> globalData gl:
63
64 FROM fca:
65 openDescriptor: DESIGNATOR:
66 openModes: {omRead, omWrite, omExclusive};
67 VFUN FCAopenEntry(seid pSeid; openDescriptor od) -> openFileEntry oe:
68
69
70 FUNCTIONS
71
72 VFUN LEVgetObjectLevel(seid pSeid, oSeid) -> tiiStruct otii,
73 EXCEPTIONS
74 EXCEPTIONS_OF TIIgetEntityLevel(pSeid, oSeid);
75 DERIVATION
76 TIIgetEntityLevel(pSeid, oSeid):
77
78 OFUN LEVsetObjectLevel(seid pSeid, oSeid; tiiStruct otii);
79                                     $(LEVsetObjectLevel)
80 DEFINITIONS
81 secureEntityType type IS SENseidType(oSeid):
82 EXCEPTIONS
83 EXCEPTIONS_OF TIIsetEntityLevel(pSeid, oSeid, otii):
84 KElevSegEx: type = tSegment AND SEGinstanceInfo(oSeid).sharable = TRUE:
85 KElevFileEx:
86     type INSET {tFile, tDevice, tSubtype, tTerminal, tExtent}
87     AND (EXISTS seid pSedil: openDescriptor od
88         : FCAopenEntry(pSeid, od).openSeid = oSeid);
89 EFFECTS
90 EFFECTS_OF TIIsetEntityLevel(pSeid, oSeid, otii);
91
92
93 END_MODULE
```

```
1 $("      MODULE:      mac.specs (version 2.4)
2          CONTENTS:    Machine
3          TYPE:        SPECIAL.specifications
4          LAST CHANGED: 6/24/79 19:31:12
5      ")
6
7
8 MODULE mac
9
10     PARAMETERS
11
12     INTEGER MACmaxVAddr, $( maximum virtual address, also maximum segment size
13                             2^16 - 1 on PDP-11/70)
14     MACmaxOffset, $( maximum offset component of virtual address,
15                     2^13 - 1 on PDP-11/70)
16     MACmaxReg: $( maximum memory mapping register address, seven on
17                  PDP-11/70)
18
19
20     ASSERTIONS
21
22     MACmaxVAddr > 0; MACmaxOffset > 0; MACmaxReg > 0;
23     MACmaxVAddr + 1 = (MACmaxOffset + 1) * (MACmaxReg + 1);
24
25
26     FUNCTIONS
27
28     VFUN MACclock() -> INTEGER time:
29         $( integer that represents real time)
30     INITIALLY TRUE:
31
32     OFUN MACclockIncrement():
33         $( invoked continuously by a separate abstract process — the system clock)
34     EFFECTS
35         'MACclock() = MACclock() + 1;
36
37
38 END MODULE
```

```

1  $( "      MODULE.      pbl.specs (version 2.7)
2      CONTENTS.      Parameter Block Functions
3      TYPE.      SPECIAL specifications
4      LAST CHANGED:  10/12/79, 14:18:28
5  ")
6
7
8  MODULE pbl
9
10 $( This module specifies the action of getting arguments or putting results
11    of operations into the virtual memory of a process. The part of
12    virtual memory so manipulated is called a parameter block. The need
13    for parameter blocks comes about when the length of the data is not
14    constant for all invocations of a given operation.
15
16    To make specifications more readable, all parameter block operations are
17    specified in two parts. The basic functionality of an operation is
18    specified in the module to whose object the operation refers, e.g.,
19    the basic specification of readBlock comes from the fmi module. The
20    parameter block manipulation, along with appropriate data conversion,
21    is specified here. This decomposition removes the issue of parameter
22    blocks from the basic specification of already complicated operations.
23    The parameter block manipulation becomes simple once isolated here.)
24
25
26    TYPES
27
28    $(from mac)
29    vAddrType: {0 .. MACmaxVAddr};
30
31    $(from pvm)
32    virtualLocation:
33    STRUCT_OF(domainType domain: spaceType idSpace: vAddrType vAddr);
34    pBlock: STRUCT_OF(virtualLocation vloc; INTEGER size);
35
36    $(from fca)
37    asyncId: CHAR;
38    fileStatus: STRUCT_OF(INTEGER nBlocks, linkCount, timeLastMod; seid subtype:
39        BOOLEAN openAtCrash);
40    ioStatus: STRUCT_OF(INTEGER devDep, devInd);
41    fileBlock: VECTOR_OF CHAR;
42    readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);
43
44    $(from spf)
45    SPFargs: VECTOR_OF INTEGER;
46
47
48    EXTERNALREFS
49
50    FROM mac:
51    INTEGER MACmaxVAddr;
52
53    FROM smx:
54    seid: DESIGNATOR;
55    domainType: {userDomain, supervisorDomain};
56

```

```

57 FROM pvm:
58 spaceType: {ispace. dSpace};
59 OFUN PVMstore(seid pSeid; pBlock block: VECTOR_OF INTEGER vec);
60 VFUN PVMretrieve(seid pSeid; pBlock block) -> VECTOR_OF INTEGER vec;
61
62 FROM fca:
63 openDescriptor: DESIGNATOR;
64 IOfunction: {rewind, etc};
65
66 FROM fmi:
67 VFUN FCAvDeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
68                          VECTOR_OF INTEGER args; asyncId id)
69   -> ioStatus status;
70 OFUN FCAdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
71                        VECTOR_OF INTEGER args; asyncId id)
72   -> ioStatus status;
73 VFUN FCAvReadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
74                     asyncId as) -> readResult rr;
75 OFUN FCAreadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
76                    asyncId as) -> readResult rr;
77 OFUN FCAwriteBlocks(seid pSeid; openDescriptor od; INTEGER blockNo;
78                     VECTOR_OF fileBlock vfb; asyncId id) -> ioStatus ios;
79
80 FROM spf:
81 SPFFunctionType: {syncSPF. immSegSPF, sysHaltSPF, levelSetSPF};
82 OFUN SPFSpecialFunction(seid pSeid; SPFFunctionType fn: SPFargs args);
83
84
85 ASSERTIONS
86
87 FORALL VECTOR_OF fileBlock vfb
88   : PBLwordsToBlocks(PBLblocksToWords(vfb)) = vfb;
89
90
91 FUNCTIONS
92
93 $(----- data conversion functions -----)
94
95 VFUN PBLioStatToVec(ioStatus ios) -> VECTOR_OF INTEGER vi;
96   HIDDEN;
97   INITIALLY vi ~= ?;
98
99 VFUN PBLblocksToWords(VECTOR_OF fileBlock vfb) -> VECTOR_OF INTEGER vi;
100   HIDDEN;
101   INITIALLY vi ~= ?;
102
103 VFUN PBLwordsToBlocks(VECTOR_OF INTEGER vi) -> VECTOR_OF fileBlock vfb;
104   HIDDEN;
105   DERIVATION SOME VECTOR_OF fileBlock vfb1 | PBLblocksToWords(vfb1) = vi;
106
107 $(----- operations -----)
108
109 OFUN PBLdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
110                        pBlock arguments, status; asyncId id);
111                        $(PBLdeviceFunction)
112
113 DEFINITIONS

```

```

113 VECTOR OF INTEGER inargs IS PVMretrieve(pSeid, arguments);
114 ioStatus st IS FCAvDeviceFunction(pSeid, od, f. inargs, id);
115 VECTOR OF INTEGER result IS PBLioStatToVec(st);
116 EXCEPTIONS
117 EXCEPTIONS OF PVMretrieve(pSeid, arguments);
118 EXCEPTIONS OF FCAvDeviceFunction(pSeid, od, f. inargs, id);
119 EXCEPTIONS OF PVMstore(pSeid, status, result);
120 EFFECTS
121 EFFECTS OF PVMstore(pSeid, status, result);
122 st = EFFECTS OF FCAvDeviceFunction(pSeid, od, f. inargs, id);
123
124 OVFUN PBLreadBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
125 pBlock duFile: asyncId as)
126 -> STRUCT OF (INTEGER bytesRead; ioStatus errst) result; $(PBLreadBlock)
127 DEFINITIONS
128 readResult rr IS FCAvReadBlocks(pSeid, od, blockNo, duFile.size, as);
129 VECTOR OF INTEGER intData IS PBLblocksToWords(rr.data);
130 EXCEPTIONS
131 EXCEPTIONS OF FCAvReadBlocks(pSeid, od, blockNo, duFile.size, as);
132 EXCEPTIONS OF PVMstore(pSeid, duFile, intData);
133 EFFECTS
134 result = STRUCT(LENGTH(intData), rr.errst);
135 EFFECTS OF PVMstore(pSeid, duFile, intData);
136
137 OFUN PBLspecialFunction(seid pSeid; SPFFunctionType fn; pBlock parm);
138 $(PBLspecialFunction)
139 DEFINITIONS
140 SPFFargs args IS PVMretrieve(pSeid, parm);
141 EXCEPTIONS
142 EXCEPTIONS OF PVMretrieve(pSeid, parm);
143 EXCEPTIONS OF SPFFspecialFunction(pSeid, fn, args);
144 EFFECTS
145 EFFECTS OF SPFFspecialFunction(pSeid, fn, args);
146
147 OVFUN PBLwriteBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
148 pBlock duFile: asyncId id) -> ioStatus ios;
149 $(PBLwriteBlock)
150 DEFINITIONS
151 VECTOR OF fileBlock vfb IS PBLwordsToBlocks(PVMretrieve(pSeid, duFile));
152 EXCEPTIONS
153 EXCEPTIONS OF PVMretrieve(pSeid, duFile);
154 EXCEPTIONS OF FCAwriteBlocks(pSeid, od, blockNo, vfb, id);
155 EFFECTS
156 ios = EFFECTS OF FCAwriteBlocks(pSeid, od, blockNo, vfb, id);
157
158 END MODULE

```

```

1  $( "      MODULE          pro.specs (version 2.11)
2          CONTENTS      Process Operators
3          TYPE          SPECIAL.specifications
4          LAST CHANGED:  7/17/79, 15:08:33
5      " )
6
7
8  MODULE pro
9
10 $( this module now contains the material which was formerly in
11    the pro, ipc and pst modules)
12
13     TYPES
14
15     $(types supporting pseudo interrupts)
16 piLevelType: {PROminPiLevel .. PROMaxPiLevel}: $(pseudo interrupt level range)
17 piEntryType: STRUCT OF(BOOLEAN pending;
18                     piLevelType oldPIL;
19                     INTEGER oldPc;
20                     INTEGER oldPs;
21                     INTEGER parameter;
22                     INTEGER newPc;
23                     INTEGER newPs);
24 piVectorType: {VECTOR OF piEntryType piv |
25                LENGTH(piv) = PROMaxPiLevel-PROminPiLevel};
26
27     $(types supporting ipc)
28 ipcqType: {VECTOR OF ipcMessageType zz | LENGTH(zz) <= IPCmaxMessageCount};
29 ipcMessageType: STRUCT OF(seid sender; ipcTextType text);
30 ipcTextType: {VECTOR OF CHAR vc | LENGTH(vc) = IPCmaxMessageLength};
31 pendingType: STRUCT OF(BOOLEAN flag; INTEGER time);
32
33     $(structure of process status information)
34 processStateType: STRUCT OF(seid self;
35                     seid parent;
36                     INTEGER family;
37                     INTEGER realUser;
38                     INTEGER realGroup;
39                     INTEGER pc;          $(program counter)
40                     INTEGER ps;          $(processor status)
41                     piLevelType pil;
42                     piVectorType piv;
43                     ipcqType ipcq;
44                     INTEGER advPrio;     $(advisory priority)
45                     INTEGER timerAlarm;  $(one-zero crossing => pi)
46                     INTEGER supervisorTiming;
47                     INTEGER userTiming;
48                     BOOLEAN timTog;      $(timer toggle TRUE is ON)
49
50     $(from smx)
51 nonDisType: STRUCT OF(
52     INTEGER securityLevel; SET OF securityCat securityCatS;
53     INTEGER integrityLevel; SET OF integrityCat integrityCatS);
54 daType: SET OF daMode;
55 modeStruct: STRUCT OF(daType ownerMode, groupMode, allMode);
56 tiisStruct: STRUCT OF(nonDisType nd;

```

```

57         modeStruct da; INTEGER owner, group; SET_OF privType priv);
58
59
60     PARAMETERS
61
62     INTEGER PROMaxProcessCount;
63
64     INTEGER PROMinPiLevel: $(most interruptable pseudo interrupt level)
65     INTEGER PROMaxPiLevel: $(least interruptable pseudo interrupt level)
66     INTEGER piZero, piIPC, piTimer, piSignal, piHardwareFault:
67         $(defined pseudo interrupt levels)
68
69     INTEGER IPCmaxMessageCount; $(maximum number of ipc messages per process)
70     INTEGER IPCmaxMessageLength; $(maximum number of characters per ipc message)
71     ipcMessageType timeoutMessage; $(distinguished message returned when K_receive
72         is satisfied by its timeout)
73
74     INTEGER newPc, $(program counter for process invocation and spawning,
75         probably 0)
76     newPs: $(processor state for process invocation and spawning,
77         probably 050000 octal)
78
79     seid processExampleSeid; $(any seid with the tProcess nsp)
80     piVectorType emptyPiv; $(used for state initialization)
81     ipcqType emptyIpcq; $( ... )
82
83
84     DEFINITIONS
85
86     BOOLEAN processExists(seid pSeid) IS PSTprocessState(pSeid) ~= ?;
87     seid newProcessSeid IS $(the process seid generation algorithm)
88     SENmakeSeid(processExampleSeid, SOME INTEGER i | EXISTS seid s :
89         SENseidType(s) = tProcess
90         AND SENseidIndex(s) = i
91         AND NOT processExists(s));
92     INTEGER processCount IS
93     CARDINALITY({INTEGER i | PSTprocessSlot(i) ~= ?});
94     INTEGER emptySlot IS SOME INTEGER i | i INSET {1 .. PROMaxProcessCount}
95         AND PSTprocessSlot(i) = ?;
96
97
98     EXTERNALREFS
99
100     FROM mac:
101     VFUN MACclock() -> INTEGER time;
102
103     FROM smx:
104     seid: DESIGNATOR:
105     secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment,
106         tSubtype, tExtent, tNull};
107     privType: {
108         privFileUpdateStatus,    privLink,        privLockSeg.
109         privModifyPriv,          privMount,
110         privSetFileLevel,        privSetSegProcLevel,
111         privStickySeg,           privTerminalLock,
112         privViolSimpSecurity,     privViolStarSecurity,

```

```

113     privViolSimpIntegrity,   privViolStarIntegrity,
114     privViolDiscrAccess,    privSignal,      privWalkPTable,
115     privHalt,               privKernelCall,  privViolCompartments,
116     privRealizeExecPermissions};
117 daMode: {daRead, daWrite, daExecute};
118 securityCat: DESIGNATOR;
119 integrityCat: DESIGNATOR;
120 VFUN SENseidIndex(seid s) -> INTEGER index;
121 VFUN SENseidType(seid s) -> secureEntityType t;
122 VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid rSeid;
123 VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
124 VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;
125 VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
126 VFUN TIIgetEntityLevel(seid pSeid, oSeid) -> tiiStruct otii;
127 VFUN TIIinfo(seid s) -> tiiStruct tiis;
128 OFUN TIIsetEntityLevel(seid pSeid, oSied; tiiStruct ntii);
129
130     FROM pvm:
131 segDes: DESIGNATOR;
132
133     FROM pvp:
134 OFUN PVPforkSupport(seid parent, child);
135 OFUN PVPinvokeSupport(seid pSeid, immSeid; segDes arg);
136 OFUN PVPspawnSupport(seid parent, child, imSeid; segDes arg);
137 OFUN PVPreleaseProcessSupport(seid pSeid);
138
139     FROM fca:
140 OFUN FCACreateOpenTable(seid pSeid);           $(spawn support)
141 OFUN FCACloseAll(seid pSeid);                 $(invoke support)
142
143     FROM fmi:
144 OFUN FMIforkSupport(seid parent, child);
145 OFUN FMIfreeSupport(seid pSeid);
146
147
148     ASSERTIONS
149 PROMinPiLevel <= piZero; piZero < piIPC; piIPC < piTimer; piTimer < piSignal;
150 piSignal < piHardwareFault; piHardwareFault <= PROMaxPiLevel;
151 $( ordering of defined pseudo interrupt levels)
152 SENseidType(processExampleSeid) = tProcess;
153 $(processExampleSeid is an example of a process seid)
154 FORALL INTEGER i INSET {PROMinPiLevel .. PROMaxPiLevel} :
155     emptyPiv[i] = STRUCT(FALSE.i.0.0.0.0.0); $(definition of empty piv)
156 LENGTH(emptyIpcq) = 0; $(emptyIpcq is in fact empty)
157 processCount <= PROMaxProcessCount; $(there are never too many processes)
158
159
160     FUNCTIONS
161
162 $( ----- process state functions ----- )
163
164 VFUN PSTprocessState(seid pSeid) -> processStateType ps;
165     HIDDEN;
166     INITIALLY ps = ?;
167
168 VFUN PSTprocessSlot(INTEGER n) -> seid ps;

```

```

169  HIDDEN:
170  INITIALLY ps = ?;
171
172  $( -----support for K walk_process_table -----)
173
174  VFUN PROwalkProcessTable(seid pSeid; INTEGER n) -> seid rSeid:
175    EXCEPTIONS
176      KEproNoPriv: NOT privWalkPTable INSET TIIinfo(pSeid).priv:
177      KEproBadSlot: NOT n+1 INSET {1 .. PROmaxProcessCount};
178    DERIVATION
179      PSTprocessSlot(n);
180
181  $( ----- support for K_nap -----)
182
183  VFUN PSTtimeOfNap(seid pSeid) -> INTEGER time:
184    HIDDEN:
185    INITIALLY time = 0;
186
187  OFUN PRONap(seid pSeid; INTEGER timeout);
188    DELAY WITH 'PSTtimeOfNap(pSeid) = MACclock();
189      UNTIL MACclock() >= PSTtimeOfNap(pSeid) + timeout;
190
191  $( ----support for pseudo interrupts: K_signal and K_interruptReturn----)
192
193  OFUN PROsignal(seid sender, receiver: INTEGER signalName);
194    DEFINITIONS
195      piEntryType receiversSignalEntry()
196        IS PSTprocessState(receiver).piv[piSignal];
197    EXCEPTIONS
198      KEproNoPriv: NOT SMXhasPriv(sender, privSignal);
199      KEproNoAccess: NOT (processExists(receiver)
200        AND SMXflow(sender, receiver, {daWrite, daRead}));
201      KEproUninterruptable: NOT PSTprocessState(receiver).pil > piSignal:
202        $(if fails, try once again after a short timeout)
203    ASSERTIONS
204      processExists(sender):
205    EFFECTS
206      'receiversSignalEntry().pending = TRUE;
207      'receiversSignalEntry().parameter = signalName;
208  $(PROBLEMS
209    Design requires that K_signal interrupt long kernel calls. How is this
210    this to be done?)
211
212  OFUN PROinterruptReturn(seid pSeid);
213  $(emulates rti instruction. Uses pseudo interrupt vector entry associated
214    with current level and restores pc, ps, and pil to their "old" values)
215    DEFINITIONS
216      piEntryType currentInterruptEntry
217        IS PSTprocessState(pSeid).piv[PSTprocessState(pSeid).pil];
218    EFFECTS
219      'PSTprocessState(pSeid).pc = currentInterruptEntry.oldPc;
220      'PSTprocessState(pSeid).ps = currentInterruptEntry.oldPs;
221      'PSTprocessState(pSeid).pil = currentInterruptEntry.oldPil;
222
223  $( ----- support for ipc: K_post and K_receive -----)
224

```

```

225 OFUN PROpost(seid sender, receiver; BOOLEAN postPseudoInterrupt;
226         ipcTextType text);
227 $(Detects security violation and overflow. Appends message to the tail of
228 the receivers ipcq. Posts pseudo interrupt in the receiver if requested
229 and if receiver has no pending receive)
230 DEFINITIONS
231     VECTOR OF ipcMessageType queue() IS PSTprocessState(receiver).ipcq;
232     INTEGER qLength IS LENGTH(queue());
233 EXCEPTIONS
234     KProNoReceiver: NOT (processExists(receiver)
235         AND SMXflow(sender, receiver, {daWrite}));
236     KProIpcOverflow: (1+qLength) > IPCmaxMessageCount;
237 ASSERTIONS
238     processExists(sender);
239     LENGTH(text) <= IPCmaxMessageLength;
240 EFFECTS
241     $(append new message to receivers queue)
242     'queue()=VECTOR (FOR i FROM 1 TO 1+qLength :
243         IF i <= qLength THEN queue()[i] ELSE STRUCT(sender, text));
244     $(post ipc pseudo interrupt if required and if receiver has no pending
245 read)
246     postPseudoInterrupt AND NOT PSTreceivePending(receiver).flag
247     => 'PSTprocessState(receiver).piv[pIPC].pending = TRUE;
248
249 VFUN PSTreceivePending(seid pSeid) -> pendingType r;
250 HIDDEN;
251 INITIALLY r.flag = FALSE AND r.time = 0;
252
253 OVFUN PROreceive(seid pSeid; INTEGER timeout) -> ipcMessageType msg;
254 $(Returns the ipc message at the head of the queue if one exists or arrives
255 before the expiration of timeout. Otherwise returns a distinguished ipc
256 message signifying timeout)
257 DEFINITIONS
258     VECTOR OF ipcMessageType queue() IS PSTprocessState(pSeid).ipcq;
259     INTEGER qLength IS LENGTH(queue());
260     pendingType pending() IS PSTreceivePending(pSeid);
261 DELAY WITH 'pending().flag = TRUE; 'pending().time = MACclock();
262 UNTIL qLength > 0 OR MACclock() >= pending().time + timeout;
263 EFFECTS
264     IF qLength > 0 THEN msg = queue()[1] ELSE msg = timeoutMessage;
265     'queue() = VECTOR( FOR i FROM 1 TO qLength-1 : queue()[i+1]);
266     'pending().flag = FALSE;
267
268 $( ----- support for K_fork -----)
269
270 OVFUN PROfork(seid parent) -> seid child;
271 DEFINITIONS
272     processStateType p IS PSTprocessState(parent);
273     seid c IS newProcessSeid;
274 EXCEPTIONS
275     KProTooManyProcesses: processCount+1 > PROmaxProcessCount;
276     EXCEPTIONS_OF FMIforkSupport(parent, c);
277     EXCEPTIONS_OF PVPforkSupport(parent, c);
278 EFFECTS
279     child = c;
280     'PSTprocessSlot(emptySlot) = child;

```

```

281 'PSTprocessState(child) = STRUCT(child, parent, p.family,
282   p.realUser, p.realGroup, p.pc, p.ps, PROMaxPiLevel,
283   emptyPiv, emptyIpcq, p.advPrio, 0, 0, 0, FALSE);
284 $(this assertion specifies the initial tdi state of a forked child)
285 'TIIinfo(child) = TIIinfo(parent);
286 $(this assertion specifies the initial tii state of a forked child)
287 EFFECTS OF FMiforkSupport(parent, child); $(provide and copy pofv)
288 EFFECTS OF PVPforkSupport(parent, child); $(provide virtual memory)
289
290 $( ----- support for K invoke -----)
291
292 OFUN PROinvoke(seid pSeid, immSeid; segDes arg);
293   DEFINITIONS
294     processStateType p IS PSTprocessState(pSeid);
295     tiiStruct pt IS TIIinfo(pSeid);
296   EXCEPTIONS
297     EXCEPTIONS OF PVPinvokeSupport(pSeid, immSeid, arg);
298   ASSERTIONS
299     processExists(pSeid);
300   EFFECTS
301     'PSTprocessState(pSeid) = STRUCT(p.self, p.parent, p.family,
302       p.realUser, p.realGroup, newPc, newPs,
303       p.pil, emptyPiv, emptyIpcq, p.advPrio, 0, 0, 0, FALSE);
304     $(assertion defines the initial process state of the invoked intermed.)
305     'TIIinfo(pSeid) = STRUCT(pt.nd, pt.da, pt.owner, pt.group,
306       TIIinfo(immSeid).priv);
307     $(this is how the post-invoke process gets the intermeds privileges)
308     EFFECTS OF PVPinvokeSupport(pSeid, immSeid, arg); $(redo virtual memory)
309
310 $( ----- support for K_spawn -----)
311
312 OVFUN PROspawn(seid parent, immSeid; segDes arg) -> seid child;
313   DEFINITIONS
314     processStateType p IS PSTprocessState(parent);
315     tiiStruct pt IS TIIinfo(parent);
316     seid c IS newProcessSeid;
317   EXCEPTIONS
318     KEproTooManyProcesses: processCount+1 > PROMaxProcessCount:
319     EXCEPTIONS OF PVPspawnSupport(parent, c, immSeid, arg);
320     EXCEPTIONS OF FCAcreateOpenTable(parent);
321   ASSERTIONS
322     processExists(parent);
323   EFFECTS
324     child = c;
325     'PSTprocessSlot(emptySlot) = child;
326     'PSTprocessState(child) = STRUCT(parent, child, p.family,
327       p.realUser, p.realGroup, newPc, newPs, PROMaxPiLevel,
328       emptyPiv, emptyIpcq, p.advPrio, 0, 0, 0, FALSE);
329     $(the process state of the newly spawned intermediary)
330     'TIIinfo(child) = STRUCT(pt.nd, pt.da, pt.owner, pt.group,
331       TIIinfo(immSeid).priv);
332     $(post-spawn child acquires intermediatarys privileges)
333     EFFECTS OF PVPspawnSupport(parent, child, immSeid, arg); $(create vm)
334     EFFECTS OF FCAcreateOpenTable(child); $(create pofv)
335
336 $( ----- support for K_release process -----)

```

```

337
338 OFUN PROreleaseProcess(seid pSeid, rSeid):
339 $(Typically a process will release itself and pSeid=rSeid. However this
340 is not treated as a special case.)
341 EXCEPTIONS
342 KEproNoRelease: NOT processExists(rSeid)
343                      OR NOT (TIIinfo(rSeid).owner = TIIinfo(pSeid).owner
344                      OR privSetSegProcLevel INSET TIIinfo(pSeid).priv):
345 ASSERTIONS
346     processExists(pSeid);
347 EFFECTS
348     'PSTprocessSlot(SOME INTEGER i | PSTprocessSlot(i) = rSeid) = ?;
349     'PSTprocessState(rSeid) = ?;
350     'TIIinfo(rSeid) = ?;
351 EFFECTS OF FMireleaseSupport(rSeid);
352 EFFECTS OF PVPreleaseProcessSupport(rSeid);
353
354 $( ----- status getting and setting ----- )
355
356 VFUN PROgetProcessStatus(seid pSeid, oSeid) -> processStateType ps;
357 EXCEPTIONS
358     KEproNoProcess: NOT processExists(oSeid)
359                      OR NOT SMXflow(pSeid, oSeid, {daRead});
360 ASSERTIONS
361     processExists(pSeid);
362 DERIVATION
363     PSTprocessState(oSeid);
364
365 OFUN PROsetProcessStatus(seid pSeid, oSeid; processStateType n):
366 DEFINITIONS
367     processStateType o IS PSTprocessState(oSeid);
368 EXCEPTIONS
369     KEproNoProcess: NOT processExists(oSeid)
370                      OR NOT SMXflow(pSeid, oSeid, {daWrite});
371 ASSERTIONS
372     processExists(pSeid);
373 EFFECTS
374     'PSTprocessState(oSeid) = STRUCT(o.self, o.parent, n.family,
375     n.realUser, n.realGroup, n.pc, n.ps, n.pil, n.piv, n.ipcq,
376     n.advPrio, n.timerAlarm, o.supervisorTiming, o.userTiming,
377     n.timTog);
378
379
380 END_MODULE

```

```

1  $( "  MODULE:      pvm.specs (version 2.37)
2      CONTENTS:     Virtual Memory
3      TYPE:         SPECIAL.specifications
4      LAST CHANGED: 10/12/79, 10:30:48
5  " )
6
7
8  MODULE pvm
9
10 $( this module now contains the contents of what was the seg and pvm modules)
11
12     TYPES
13
14     $(FROM mac)
15 vAddrType: {0 .. MACmaxVAddr};
16
17     $(FROM smx)
18 daType: SET OF daMode;
19 modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
20 nonDisType: STRUCT_OF(
21     INTEGER securityLevel; SET OF securityCat securityCatS;
22     INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
23 ttiStruct: STRUCT_OF(nonDisType nd;
24     modeStruct da; INTEGER owner, group; SET_OF privType priv);
25
26     $(from pvm -- exportable)
27 segDes: DESIGNATOR;
28 spaceType: {iSpace, dSpace};
29 direction: {up, down};
30
31     $(from pvm -- redeclarable)
32 virtualLocation:
33     STRUCT_OF(domainType domain; spaceType idSpace; vAddrType vAddr);
34 pBlock: STRUCT_OF(virtualLocation vloc; INTEGER size);
35 globalData:
36     STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
37     direction growth);
38 statusStruct:
39     STRUCT_OF(globalData gl; INTEGER size);
40 instanceStruct:
41     STRUCT_OF(globalData gl; INTEGER refCount; VECTOR_OF INTEGER data);
42 useStruct: STRUCT_OF(seid instance; virtualLocation vloc; daType da);
43
44
45     PARAMETERS
46
47     $(from seg)
48 seid exampleSegmentSeid; $(used for segment creation)
49 segDes SEGnullSeg; $( indicates null segment designator)
50 INTEGER PVMmaxSegDes; $(maximum number of segment designators in an address
51     space)
52
53
54     DEFINITIONS
55
56     $(from seg)

```

```

57 INTEGER SEGsize(seid segSeid) IS LENGTH(SEGinstanceInfo(segSeid).data):
58
59     $(from pvm)
60 INTEGER nSegs(seid pSeid)
61     IS CARDINALITY({segDes sd | SEGuseInfo(pSeid, sd) ~= ?}).
62
63 BOOLEAN PVMvmExists(seid pSeid) IS PVMsegmentSet(pSeid) ~= ?;
64
65 segDes PVMblockToSeg(seid pSeid; pBlock block) IS
66     SOME segDes sd
67         | sd INSET PVMmappedSegmentSet(pSeid)
68             AND (EXISTS useStruct use = SEGuseInfo(pSeid, sd)
69                 : use.vloc.domain = block.vloc.domain
70                 AND use.vloc.idSpace = block.vloc.idSpace
71                 AND use.vloc.vAddr <= block.vloc.vAddr
72                 AND block.vloc.vAddr + block.size
73                     <= use.vloc.vAddr + SEGsize(use.instance));
74     $( gives the segment designator, if any, that totally contains block in
75       address space designated by pSeid; if there is none, returns ?; the
76       segment must be mapped)
77
78 SET OF INTEGER addrRegRange(INTEGER vAddr, size; direction d) IS
79     IF d = up
80         THEN {vAddr / MACmaxOffset .. (vAddr + size - 1) / MACmaxOffset}
81         ELSE {(vAddr - size + 1) / MACmaxOffset .. vAddr / MACmaxOffset};
82     $( gives the range of address registers used by a segment as a function of
83       its start address, size, and growth direction)
84
85 SET OF INTEGER addrRegRangeSeg(seid pSeid; segDes s) IS
86     LET useStruct use = SEGuseInfo(pSeid, s)
87     IN addrRegRange(use.vloc.vAddr, SEGsize(use.instance),
88                     SEGinstanceInfo(use.instance).gl.growth);
89     $( gives the range of address registers used by a segment as a function of
90       the process id and the segment designator)
91
92 BOOLEAN noHole(seid pSeid; INTEGER size; virtualLocation vl; direction d:
93     SET OF segDes ssd) IS
94     NOT addrRegRange(vl.vAddr, size, d) SUBSET {0 .. MACmaxReg}
95     OR (EXISTS segDes s | s INSET ssd: useStruct use = SEGuseInfo(pSeid, s)
96         : use.vloc.idSpace = vl.idSpace
97         AND use.vloc.domain = vl.domain
98         AND addrRegRangeSeg(pSeid, s)
99         INTER addrRegRange(vl.vAddr, size, d) ~= {});
100     $(TRUE iff a segment described by size, vl, and direction will NOT fit into
101     a hole in the address space designated by pSeid and ssd; this includes
102     testing for virtual memory underflow and overflow)
103
104     EXTERNALREFS
105
106     FROM mac:
107     INTEGER MACmaxVAddr, MACmaxOffset, MACmaxReg;
108
109     FROM smx:
110     seid: DESIGNATOR;
111     secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
112         tExtent, tNull};

```

```

113 privType: {privFileUpdateStatus,   privLink,       privLockSeg,
114             privModifyPriv          privMount,
115             privSetFileLevel        privSetSegProcLevel,
116             privStickySeg,          privTerminalLock,
117             privViolSimpSecurity,    privViolStarSecurity,
118             privViolSimpIntegrity,   privViolStarIntegrity,
119             privViolDiscrAccess,     privSignal,     privWalkPTable,
120             privHalt,                privKernelCall, privViolCompartments,
121             privRealizeExecPermissions};
122 daMode: {daRead, daWrite, daExecute};
123 securityCat: DESIGNATOR;
124 integrityCat: DESIGNATOR;
125 domainType: {userDomain, supervisorDomain};
126 VFUN SENseidNsp(seid s) -> INTEGER nsp;
127 VFUN SENseidType(seid s) -> secureEntityType set;
128 VFUN TIInfo(seid anySeid) -> tiStruct tiSt;
129 VFUN TIInfoGetEntityLevel(seid pSeid, oSeid) -> tiStruct otii;
130 OFUN TIInfoSetEntityLevel(seid pSeid, oSeid; tiStruct ntii);
131 VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
132 VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;
133 VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
134
135
136     ASSERTIONS
137
138     $(from seg)
139     PVMmaxSegDes >= 2;
140     $( basic relations among the SEG parameters)
141     SENseidType(exampleSegmentSeid) = tSegment;
142     $( basic property of exampleSegmentSeid and all segment seids)
143     FORALL seid s | SEGinstanceInfo(s) ~= ?
144       : SENseidNsp(s) = SENseidNsp(exampleSegmentSeid);
145     $(all seids for existing segments have a distinguished nsp component)
146     FORALL seid s | SEGinstanceInfo(s) ~= ? : TIInfo(s) ~= ?;
147     $(all existing segments have an exiting TII entry)
148     FORALL seid pSeid; segDes segd | SEGuseInfo(pSeid, segd) ~= ?
149       : SEGinstanceInfo(SEGuseInfo(pSeid, segd).instance) ~= ?;
150     $(all valid segment uses have corresponding valid segment instances)
151     FORALL seid s | SEGinstanceInfo(s) ~= ?
152       : LET modeStruct ms = TIInfo(s).da
153         IN (daWrite INSET ms.ownerMode => daRead INSET ms.ownerMode)
154           AND (daWrite INSET ms.groupMode => daRead INSET ms.groupMode)
155           AND (daWrite INSET ms.allMode => daRead INSET ms.allMode);
156     $(write access implies read access, because the hardware does not support
157       write-only access for segments)
158     FORALL seid pSeid; segDes sd | SEGuseInfo(pSeid, sd) ~= ?
159       : daWrite INSET SEGuseInfo(pSeid, sd).da
160         => daRead INSET SEGuseInfo(pSeid, sd).da;
161     $(same constraint as above, for segment use information)
162
163     $(from pvm)
164     FORALL seid pSeid | PVMvmExists(pSeid); segDes sd
165       : (sd INSET PVMsegmentSet(pSeid)) = (SEGuseInfo(pSeid, sd) ~= ?);
166     $(defines what it means for a segment to be in the segment set of a
167       process)
168     FORALL seid pSeid | PVMvmExists(pSeid)

```

```

169 PVMmappedSegmentSet(pSeid) SUBSET PVMsegmentSet(pSeid):
170 $(only existing segments can be mapped)
171 FORALL seid pSeid | PVMvmExists(pSeid): segDes s1, s2
172 : LET useStruct use1 = SEGuseInfo(pSeid, s1);
173     useStruct use2 = SEGuseInfo(pSeid, s2)
174     IN s1 ~= s2 AND use1 ~= ? AND use2 ~= ?
175         AND {s1, s2} SUBSET PVMmappedSegmentSet(pSeid)
176         AND use1.vloc.domain = use2.vloc.domain
177         AND use1.vloc.idSpace = use2.vloc.idSpace
178     => addrRegRangeSeg(pSeid, s1) INTER addrRegRangeSeg(pSeid, s2) = {};
179 $(no two mapped segments in the same domain and idSpace may have
180     overlapping memory address registers)
181
182
183 FUNCTIONS
184
185 $(----- state functions -----)
186
187 VFUN SEGinstanceInfo(seid segSeid) -> instanceStruct is: $(SEGinstanceInfo)
188 $( gives all the information pertaining to a segment's global data,
189     referred to as segment instance data)
190 HIDDEN;
191 INITIALLY is = ?;
192
193 VFUN SEGuseInfo(seid pSeid: segDes segd) -> useStruct us: $(SEGuseInfo)
194 $( gives all the information pertaining to a segment's use in the address
195     space in a particular process; this is information local to a process)
196 HIDDEN;
197 INITIALLY us = ?;
198
199 VFUN PVMsegmentSet(seid pSeid) -> SET_OF segDes segSet; $(PVMsegmentSet)
200 $( gives the set of segments possessed by a given process)
201 INITIALLY segSet = ?;
202
203 VFUN PVMmappedSegmentSet(seid pSeid) -> SET_OF segDes mappedSet;
204                                     $(PVMmappedSegmentSet)
205 $( gives the set of mapped — or active segments — of a process; a
206     segment cannot be addressed unless it is mapped)
207 HIDDEN;
208 INITIALLY mappedSet = ?;
209
210 $(----- virtual memory management -----)
211
212 OFUN PVMcreateVM(seid pSeid); $(PVMcreateVM)
213 $( creates a new virtual memory to be identified by "pSeid": this VM
214     must not currently exist)
215 ASSERTIONS
216     NOT PVMvmExists(pSeid);
217 EFFECTS
218     'PVMsegmentSet(pSeid) = {};
219     'PVMmappedSegmentSet(pSeid) = {};
220
221 OFUN PVMdeleteVM(seid pSeid); $(PVMdeleteVM)
222 $( deletes the currently existing virtual memory "pSeid")
223 ASSERTIONS
224     PVMvmExists(pSeid);

```

```

225 EFFECTS
226   'PVMsegmentSet(pSeid) = ?;
227   'PVMmappedSegmentSet(pSeid) = ?;
228
229 $(----- basic segment management -----)
230
231 OFUN PVMstore(seid pSeid: pBlock block: VECTOR OF INTEGER vec); $(PVMstore)
232 $(inserts contents of vec into the mapped segment indicated by block)
233 DEFINITIONS
234   segDes targ IS PVMblockToSeg(pSeid, block);
235   useStruct use IS SEGuseInfo(pSeid, targ);
236   instanceStruct inst IS SEGinstanceInfo(use.instance);
237 EXCEPTIONS
238   KEpvmVecTooLong: LENGTH(vec) ~= block.size;
239   KEpvmNoSuchSeg: targ = ?;
240   $(there is a single segment in the address space of "pSeid" in which
241     "pBlock" fits, having the same domain and idSpace of pBlock)
242   KEpvmNotWritable: NOT SMXdap(pSeid, use.instance, {daWrite});
243 ASSERTIONS
244   PVMvmExists(pSeid);
245 EFFECTS
246   LET INTEGER relOffset = block.vloc.vAddr - use.vloc.vAddr
247   IN 'SEGinstanceInfo(use.instance) =
248     STRUCT(inst.gl, inst.refCount,
249     VECTOR(
250       FOR i FROM 1 TO LENGTH(inst.data)
251       : IF i INSET {relOffset + 1 .. relOffset + LENGTH(vec)}
252       THEN vec[i - relOffset]
253       ELSE inst.data[i]));
254
255 VFUN PVMretrieve(seid pSeid: pBlock block) -> VECTOR OF INTEGER vec;
256 $( retrieves data from a mapped segment as specified by pBlock)
257 DEFINITIONS
258   segDes targ IS PVMblockToSeg(pSeid, block);
259   useStruct use IS SEGuseInfo(pSeid, targ);
260   instanceStruct inst IS SEGinstanceInfo(use.instance);
261 EXCEPTIONS
262   KEpvmNoSuchSeg: targ = ?;
263   $(there is a single segment in the address space of "pSeid" in which
264     "pBlock" fits, having the same domain and idSpace of pBlock)
265   KEpvmNotReadable: NOT SMXflow(pSeid, use.instance, {daRead});
266 ASSERTIONS
267   PVMvmExists(pSeid);
268 DERIVATION
269   VECTOR(FOR i FROM 1 TO block.size:
270     inst.data[block.vloc.vAddr - use.vloc.vAddr + i - 1]);
271
272 OVFUN PVMbuild(seid pSeid: statusStruct st: modeStruct ms; INTEGER size;
273   virtualLocation vl)
274 -> STRUCT OF(seid segSeid: segDes segd) result; $(PVMbuild)
275 $( builds a new segment with the specified parameters:
276   an entry in the tii table is also created for the segment;
277   the results are the -- previously unused -- seid for the new segment and
278   the -- previously unused -- segment designator: the newly created
279   segment is mapped, indicating that it can be addressed immediately;
280   discretionary access that the process allows itself to the segment, "da"

```

```

281 must be a subset of the owner access specified in the tii information
282 "ms" the new segment must be within the size limitations and fit into
283 the mapped virtual memory space of the creating process)
284 DEFINITIONS
285 tiiStruct proTii IS TIIinfo(pSeid);
286 tiiStruct segTii
287 IS STRUCT(proTii.nd, ms, proTii.owner, proTii.group, proTii.priv);
288 seid newSegSeid
289 IS SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
290 AND SEGinstanceInfo(s) = ?;
291 EXCEPTIONS
292 KSegSticky: st.gl.sticky AND NOT SMXhasPriv(pSeid, privStickySeg);
293 KSegSwappable:
294 NOT st.gl.swappable AND NOT SMXhasPriv(pSeid, privLockSeg);
295 KSegBadMode:
296 (daWrite INSET ms.ownerMode AND NOT daRead INSET ms.ownerMode)
297 OR (daWrite INSET ms.groupMode AND NOT daRead INSET ms.groupMode)
298 OR (daWrite INSET ms.allMode AND NOT daRead INSET ms.allMode);
299 KSegBadSize: NOT size INSET {0 .. MACmaxVAddr - 1};
300 KEpvmNoHole:
301 noHole(pSeid, st.size, vl, st.gl.growth, PVMmappedSegmentSet(pSeid));
302 KEpvmNoDes: nSegs(pSeid) >= PVMmaxSegDes;
303 RESOURCE ERROR: $( ran out of table space or seid space)
304 ASSERTIONS
305 PVMvmExists(pSeid);
306 EFFECTS
307 'TIIinfo(newSegSeid) = segTii;
308 'SEGinstanceInfo(newSegSeid) =
309 STRUCT(st.gl, l,
310 VECTOR(FOR i FROM 1 TO size : 0));
311 EXISTS segDes sd | SEGuseInfo(pSeid, sd).instance = ?
312 : 'SEGuseInfo(pSeid, sd) = STRUCT(newSegSeid, vl, ms.ownerMode)
313 AND result = STRUCT(newSegSeid, sd)
314 AND 'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {sd}
315 AND 'PVMmappedSegmentSet(pSeid)
316 = PVMmappedSegmentSet(pSeid) UNION {sd};
317
318 OFUN PVMdestroy(seid pSeid; segDes segd); $(PVMdestroy)
319 $(destroys the segment use indicated by pSeid and segd; if the segment is
320 unsticky and otherwise unreferenced, the segment instance information is
321 also deleted)
322 DEFINITIONS
323 useStruct use IS SEGuseInfo(pSeid, segd);
324 seid segSeid IS use.instance;
325 instanceStruct inst IS SEGinstanceInfo(segSeid);
326 EXCEPTIONS
327 KSegNotHeld: NOT segd INSET PVMsegmentSet(pSeid);
328 ASSERTIONS
329 PVMvmExists(pSeid);
330 EFFECTS
331 'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) DIFF {segd};
332 'PVMmappedSegmentSet(pSeid) = PVMmappedSegmentSet(pSeid) DIFF {segd};
333 'SEGuseInfo(pSeid, segd) = ?;
334 IF (inst.refCount = 1 AND inst.gl.sticky = FALSE)
335 THEN 'SEGinstanceInfo(segSeid) = ?
336 AND 'TIIinfo(segSeid) = ?

```



```

393 $(----- segment sharing -----)
394
395 OVFUN PVMrendezvous(seid pSeid, segSeid; virtualLocation vl; daType da)
396   -> segDes sd; $(PVMrendezvous)
397   $( creates a use for the segment named by segSeid; this segment appears
398     inaccessible if the multilevel security model would consider it a
399     violation of information flow)
400   DEFINITIONS
401     instanceStruct inst IS SEGinstanceInfo(segSeid);
402   EXCEPTIONS
403     KEpvmWriteOnly: daWrite INSET da AND NOT daRead INSET da;
404     KSegBadName: inst = ? OR NOT SMXflow(pSeid, segSeid, da);
405     KEpvmNoDa: NOT SMXdap(pSeid, segSeid, da);
406     KEpvmDupSeg: EXISTS segDes sdl
407       : SEGuseInfo(pSeid, sdl).instance = segSeid;
408     KEpvmNoHole: noHole(pSeid, SEGsize(segSeid), vl, inst.gl.growth,
409       PVMmappedSegmentSet(pSeid));
410     KEpvmNoDes: nSegs(pSeid) >= PVMmaxSegDes;
411   ASSERTIONS
412     PVMvmExists(pSeid);
413   EFFECTS
414     LET segDes segd | SEGuseInfo(pSeid, segd) = ?
415     IN 'SEGuseInfo(pSeid, segd) = STRUCT(segSeid, vl, da)
416       AND 'SEGinstanceInfo(segSeid) =
417         STRUCT(inst.gl, inst.refCount + 1, inst.data)
418       AND 'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {segd}
419       AND 'PVMmappedSegmentSet(pSeid)
420         = PVMmappedSegmentSet(pSeid) UNION {segd}
421       AND sd = segd;
422
423 OFUN PVMcopySeg(seid fromSeid, toSeid; segDes sd): $(PVMcopySeg)
424   $( copies a segment from the virtual memory "fromSeid" to the virtual
425     memory "toSeid"; both virtual memories must exist; the segment
426     designator sd must exist in "fromSeid" but not in "toSeid"; used by
427     the module that sets up virtual memories for new processes)
428   DEFINITIONS
429     useStruct use IS SEGuseInfo(fromSeid, sd);
430     seid oldSeid IS use.instance;
431     instanceStruct inst IS SEGinstanceInfo(oldSeid);
432     seid newSeid
433       IS SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
434         AND SEGinstanceInfo(s) = ?;
435     tiiStruct stii IS TIIinfo(oldSeid);
436     tiiStruct ptii IS TIIinfo(toSeid);
437   ASSERTIONS
438     PVMvmExists(fromSeid);
439     PVMvmExists(toSeid);
440     sd INSET PVMsegmentSet(fromSeid);
441     NOT sd INSET PVMsegmentSet(toSeid);
442   EFFECTS
443     'SEGinstanceInfo(newSeid) = STRUCT(inst.gl, 1, inst.data);
444     'SEGuseInfo(toSeid, sd) = STRUCT(newSeid, use.vloc, use.da);
445     'TIIinfo(newSeid) = STRUCT(stii.nd, stii.da, ptii.owner, ptii.group,
446       stii.priv);
447     'PVMsegmentSet(toSeid) = PVMsegmentSet(toSeid) UNION {sd};
448     sd INSET PVMmappedSegmentSet(fromSeid) =>

```

```

449     'PVMmappedSegmentSet(toSeid) = PVMmappedSegmentSet(toSeid) UNION {sd}:
450
451 $(----- segment status manipulation -----)
452
453 VFUN PVMgetSegmentStatus(seid pSeid, segSeid) -> statusStruct ss;
454                                     $(PVMgetSegmentStatus)
455 $( returns the status information -- which is much of the global
456    information -- for the segment; the segment must exist in the segment
457    set of the requesting process)
458 EXCEPTIONS
459     KEnoSeg: SEGinstanceInfo(segSeid) = ?
460             OR NOT SMXflow(pSeid, segSeid, {daRead});
461 ASSERTIONS
462     PVMvmExists(pSeid);
463 DERIVATION
464     STRUCT(SEGinstanceInfo(segSeid).gl. SEGsize(segSeid));
465
466 OFUN PVMsetSegmentStatus(seid pSeid, segSeid: globalData glo);
467                                     $(PVMsetSegmentStatus)
468 $( changes the status information for the segment; certain privileges
469    are required;)
470 DEFINITIONS
471     instanceStruct i IS SEGinstanceInfo(segSeid);
472 EXCEPTIONS
473     KEpvmNoSeg: i = ? OR NOT SMXflow(pSeid, segSeid, {daRead, daWrite});
474     KEpvmBadDa: NOT SMXdap(pSeid, segSeid, {daRead, daWrite});
475     KEpvmExecute: i.gl.sharable;
476     KEpvmBadGrowth: glo.growth ~= i.gl.growth;
477     KEpvmNoSwap:
478         glo.swappable AND NOT i.gl.swappable
479         AND NOT SMXhasPriv(pSeid, privLockSeg);
480     KEpvmNoStick:
481         glo.sticky AND NOT i.gl.sticky
482         AND NOT SMXhasPriv(pSeid, privStickySeg);
483 ASSERTIONS
484     PVMvmExists(pSeid);
485 EFFECTS
486     'SEGinstanceInfo(segSeid) = STRUCT(glo, i.refCount, i.data);
487
488
489 END MODULE

```

```

1  $( " MODULE          pvp.specs (version 2.7)
2      CONTENTS:        Virtual Memory — Process Support
3      TYPE:
4      LAST CHANGED:    7/17/79  15:09:51
5  " )
6
7
8  MODULE pvp
9
10 $( this module contains operations that support the process module's use of
11    the virtual memory mechanism. This is entirely a procedure abstraction.
12    It is specified in terms of V-functions of other modules, but is
13    implemented by a program in terms of operations of other modules.
14    This sequence will be included in the comments for each of the operations.
15    This module has no state of its own except for parameters for immediate
16    segments.)
17
18    TYPES
19
20    $(from mac)
21    vAddrType { 0 .. MACmaxVAddr},
22
23    $(from smx)
24    nonDisType: STRUCT OF(
25        INTEGER securityLevel; SET OF securityCat securityCatS;
26        INTEGER integrityLevel; SET OF integrityCat integrityCatS);
27    daType: SET OF daMode;
28    modeStruct: STRUCT OF(daType ownerMode, groupMode, allMode);
29    tiiStruct: STRUCT OF(nonDisType nd;
30        modeStruct da; INTEGER owner, group; SET OF privType priv);
31
32    $(from pvm)
33    virtualLocation:
34    STRUCT OF(domainType domain; spaceType idSpace; vAddrType vAddr);
35    globalData:
36    STRUCT OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
37        direction growth);
38    instanceStruct:
39    STRUCT OF(globalData gl; INTEGER refCount; VECTOR OF INTEGER data);
40    useStruct: STRUCT OF(seid instance; virtualLocation vloc; daType da);
41
42
43    PARAMETERS
44
45    virtualLocation PVMimmVloc; $(location for immediate segment)
46    segDes PVMimmDes; $(designator for immediate text segment)
47    segDes PVMargDes; $(designator for argument segment)
48    virtualLocation PVMargVloc; $(location for argument segment)
49
50
51    DEFINITIONS
52
53    $(from seg)
54    INTEGER SEGsize(seid segSeid) IS LENGTH(SEGinstanceInfo(segSeid).data);
55
56    $(from pvm)

```

```

57 INTEGER nSegs(seid pSeid)
58 IS CARDINALITY({segDes sd | SEGuseInfo(pSeid, sd) ~= ?}):
59
60 BOOLEAN PVMvmExists(seid pSeid) IS PVMsegmentSet(pSeid) ~= ?;
61
62 SET OF INTEGER addrRegRange(INTEGER vAddr, size; direction d) IS
63 IF d = up
64 THEN {vAddr / MACmaxOffset .. (vAddr + size - 1) / MACmaxOffset}
65 ELSE {(vAddr - size + 1) / MACmaxOffset .. vAddr / MACmaxOffset}.
66 $( gives the range of address registers used by a segment as a function of
67 its start address, size, and growth direction)
68
69 SET OF INTEGER addrRegRangeSeg(seid pSeid; segDes s) IS
70 LET useStruct use = SEGuseInfo(pSeid, s)
71 IN addrRegRange(use.vloc.vAddr, SEGsize(use.instance),
72 SEGinstanceInfo(use.instance).gl.growth):
73 $( gives the range of address registers used by a segment as a function of
74 the process id and the segment designator)
75
76 BOOLEAN noHole(seid pSeid; INTEGER size; virtualLocation vl; direction d:
77 SET OF segDes ssd) IS
78 NOT addrRegRange(vl.vAddr, size, d) SUBSET {0 .. MACmaxReg}
79 OR (EXISTS segDes s | s INSET ssd; useStruct use = SEGuseInfo(pSeid, s)
80 : use.vloc.idSpace = vl.idSpace
81 AND use.vloc.domain = vl.domain
82 AND addrRegRangeSeg(pSeid, s)
83 INTER addrRegRange(vl.vAddr, size, d) ~= {});
84 $(TRUE iff a segment described by size, vl, and direction will NOT fit into
85 a hole in the address space designated by pSeid and ssd: this includes
86 testing for virtual memory underflow and overflow)
87
88 EXTERNALREFS
89
90 FROM mac:
91 INTEGER MACmaxVAddr, MACmaxOffset, MACmaxReg;
92
93 FROM smx:
94 seid: DESIGNATOR;
95 privType: {privFileUpdateStatus, privLink, privLockSeg,
96 privModifyPriv, privMount,
97 privSetFileLevel, privSetSegProcLevel,
98 privStickySeg, privTerminalLock,
99 privViolSimpSecurity, privViolStarSecurity,
100 privViolSimpIntegrity, privViolStarIntegrity,
101 privViolDiscrAccess, privSignal, privWalkPTable,
102 privHalt, privKernelCall, privViolCompartments,
103 privRealizeExecPermissions};
104 daMode: {daRead, daWrite, daExecute};
105 securityCat: DESIGNATOR;
106 integrityCat: DESIGNATOR;
107 domainType: {userDomain, supervisorDomain};
108 VFUN SENseidNsp(seid s) -> INTEGER nsp;
109 VFUN TIInfo(seid anySeid) -> tiiStruct tiiSt;
110 VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;
111 VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
112

```

```

113 FROM pvm:
114 segDes: DESIGNATOR;
115 spaceType: {iSpace, dSpace};
116 direction: {up, down};
117 seid exampleSegmentSeid; $(used for segment creation)
118 INTEGER PVMmaxSegDes;
119 VFUN SEGinstanceInfo(seid segSeid) -> instanceStruct is;
120 VFUN SEGuseInfo(seid pSeid; segDes segd) -> useStruct us;
121 VFUN PVMsegmentSet(seid pSeid) -> SET OF segDes segSet;
122 VFUN PVMmappedSegmentSet(seid pSeid) -> SET OF segDes mappedSet;
123
124
125 ASSERTIONS
126
127 PVMimmVloc.domain = supervisorDomain;
128 PVMimmVloc.idSpace = iSpace;
129 PVMargVloc.domain = supervisorDomain;
130 PVMargVloc.idSpace = dSpace;
131 $( constraints on parameters)
132
133
134 FUNCTIONS
135
136 $(----- support for PSTreleaseProcess -----)
137
138 OFUN PVPreleaseProcessSupport(seid pSeid);
139 $( This function supports PSTreleaseProcess by deleting all segments in
140 the virtual memory named by "pSeid" and then deleting the virtual memory
141 itself)
142 ASSERTIONS
143 PVMvmExists(pSeid);
144 EFFECTS
145 FORALL segDes sd | SEGuseInfo(pSeid, sd) ~= ?
146   'SEGuseInfo(pSeid, sd) = ?
147   AND (LET seid segSeid = SEGuseInfo(pSeid, sd).instance:
148       instanceStruct inst
149       = SEGinstanceInfo(segSeid)
150       IN IF inst.refCount = 1 AND inst.gl.sticky = FALSE
151          THEN 'SEGinstanceInfo(segSeid) = ?
152              AND 'TIIinfo(segSeid) = ?
153              ELSE 'SEGinstanceInfo(segSeid)
154                  = STRUCT(inst.gl, inst.refCount - 1, inst.data));
155 'PVMsegmentSet(pSeid) = ?;
156 'PVMmappedSegmentSet(pSeid) = ?;
157
158
159 $( NOTE -- There are two special segments that are appropriate to the invoke
160 and spawn operations: the argument segment "arg", already in the virtual
161 memory, which contains the data to be used by the initialized process;
162 and the immediate segment, "immSeid", which contains the code for the
163 process -- this code is also called the process bootstrapper)
164
165 $(----- support for PSTinvoke -----)
166
167 OFUN PVPinvokeSupport(seid pSeid, immSeid; segDes arg): $(PVPinvokeSupport)
168 $( This function sets up the virtual memory of a process for

```

```

169     invocation; the new mapped set contains all previously mapped supervisor
170     segments and the argument and immediate segments: the argument segment
171     is mapped to a different virtual location, and a use is created for the
172     the immediate segment)
173 DEFINITIONS
174     instanceStruct immInst IS SEGinstanceInfo(immSeid);
175     useStruct argUse IS SEGuseInfo(pSeid, arg);
176     seid argSeid IS argUse.instance;
177     instanceStruct argInst IS SEGinstanceInfo(argSeid);
178 EXCEPTIONS
179     KEpvmNoArg: argUse = ?.
180     KEpvmArgSharable: argInst.gl.sharable;
181     KEpvmArgNotWritable: NOT daWrite INSET argUse.da;
182     KEpvmBadSeg: immInst = ?
183                 OR NOT SMXflow(pSeid, immSeid, {daRead});
184     KEpvmBadDa: NOT SMXdap(pSeid, immSeid, {daExecute});
185     KEpvmArgOverflow:
186         NOT addrRegRange(PVMargVloc.vAddr, SEGsize(argSeid), argInst.gl.growth)
187         SUBSET {0 .. MACmaxReg};
188     KEpvmImmOverflow:
189         NO addrRegRange(PVMimmVloc.vAddr, SEGsize(immSeid), immInst.gl.growth)
190         SUBSET {0 .. MACmaxReg};
191     nSegs(pSeid) >= PVMmaxSegDes;
192 ASSERTIONS
193     PVMvmExists(pSeid);
194 EFFECTS
195     'SEGuseInfo(pSeid, PVMargDes)
196     = STRUCT(argUse.instance, PVMargVloc, argUse.da);
197     $( create a reference to the immediate segment)
198     'SEGuseInfo(pSeid, PVMimmDes)
199     = STRUCT(immSeid, PVMimmVloc, {daRead, daExecute});
200     'SEGinstanceInfo(immSeid)
201     = STRUCT(immInst.gl, immInst.refCount + 1, immInst.data);
202     $( add the immediate segment to the address space)
203     'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {PVMimmDes};
204     $( unmap all segments except supervisor segments, the argument segment,
205     and the immediate segment)
206     'PVMmappedSegmentSet(pSeid)
207     = {PVMargDes, PVMimmDes}
208     UNION
209     {segDes sd
210       | sd INSET PVMmappedSegmentSet(pSeid)
211       AND SEGuseInfo(pSeid, sd).vloc.domain = supervisorDomain};
212     $( remap the argument segment)
213
214 $(-----support for PSTspawn -----)
215
216 OFUN PVPspawnSupport(seid parent, child; seid immSeid; segDes arg);
217                                     $(PVPspawnSupport)
218     $(creates a new address space named by child and inserts into it two
219     segment uses -- the argument segment, arg, which is copied from the parent
220     address space, but occupies a different position -- PVMargDes --
221     and the immediate segment, immSeid, which is shared)
222 DEFINITIONS
223     instanceStruct immInst IS SEGinstanceInfo(immSeid);
224     useStruct argUse IS SEGuseInfo(parent, arg);

```

```

225     seid argSeid IS argUse.instance;
226     instanceStruct argInst IS SEGinstanceInfo(argSeid);
227     seid argCopy
228         IS SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
229             AND SEGinstanceInfo(s) = ?;
230     tiiStruct aTii IS TIIinfo(argSeid);
231     tiiStruct pTii IS TIIinfo(parent);
232     tiiStruct nTii IS STRUCT(aTii.nd, aTii.da, pTii.owner, pTii.group,
233         aTii.priv);
234 EXCEPTIONS
235     KEsegNotNull: argInst = ?;
236     KEpvmBadSeg: immInst = ? OR NOT SMXflow(parent, immSeid, {daRead});
237     KEpvmBadDa: NOT SMXdap(parent, immSeid, {daRead, daExecute});
238     KEpvmArgSharable: argInst.gl.sharable;
239     KEpvmArgNotWritable: NOT daWrite INSET argUse.da;
240     KEpvmArgOverflow:
241         NOT addrRegRange(PVMargVloc.vAddr, SEGsize(argSeid), argInst.gl.growth)
242         SUBSET {0 .. MACmaxReg};
243     KEpvmImmOverflow:
244         NOT addrRegRange(PVMimmVloc.vAddr, SEGsize(immSeid), immInst.gl.growth)
245         SUBSET {0 .. MACmaxReg};
246     RESOURCE_ERROR;
247 ASSERTIONS
248     PVMvmExists(parent);
249     NOT PVMvmExists(child);
250 EFFECTS
251     $( create a copy of the argument segment)
252     'TIIinfo(argCopy) = nTii;
253     'SEGinstanceInfo(argCopy) = STRUCT(argInst.gl, 1, argInst.data);
254     'SEGuseInfo(child, PVMargDes) = argUse;
255     $( create a use for immSeid in child)
256     'SEGuseInfo(child, PVMimmDes)
257         = STRUCT(immSeid, PVMimmVloc, {daRead, daExecute});
258     'SEGinstanceInfo(immSeid)
259         = STRUCT(immInst.gl, immInst.refCount + 1, immInst.data);
260     'PVMsegmentSet(child) = {PVMimmDes, PVMargDes};
261     'PVMmappedSegmentSet(child) = {PVMimmDes, PVMargDes};
262
263     $(----- support for PROfork -----)
264
265     OFUN PVPforkSupport(seid parent, child);                                $(PVPforkSupport)
266     $(creates a new virtual memory, child, that is a copy of parent; some
267     segments are copied and others are merely shared; if a segment is
268     sharable in the parent process, it is not copied, but a use corresponding
269     to the instance in parent is created instead; if the segment is not
270     sharable, then a new instance of the segment is created, requiring the
271     allocation of an unused seid; in either case, corresponding segments have
272     identical segment designators in both processes; much mechanism in this
273     specification is devoted to describing the set of new seids created and
274     the mapping of this set onto the set of new segment instances)
275 DEFINITIONS
276     INTEGER nCopies
277     IS CARDINALITY
278     ({segDes sd | SEGinstanceInfo(SEGuseInfo(parent,
279         sd).instance).gl.sharable
280         = FALSE});

```

```

281      $(number of nonsharable segments in parent process)
282      SET OF seid copySet
283      IS SOME SET OF seid ss
284          | CARDINALITY(ss) = nCopies
285          AND (FORALL seid s INSET ss
286              : SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
287              AND SEGinstanceInfo(s) = ?);
288      $(actual set of new seids)
289      EXCEPTIONS
290          RESOURCE ERROR;
291      ASSERTIONS
292          PVMvmExists(parent).
293          NOT PVMvmExists(child);
294      EFFECTS
295          'PVMsegmentSet(child) = PVMsegmentSet(parent);
296          'PVMmappedSegmentSet(child) = PVMmappedSegmentSet(parent);
297      FORALL segDes segd | SEGuseInfo(parent, segd).instance ~= ?
298          : LET useStruct use = SEGuseInfo(parent, segd);
299              seid segSeid = use.instance;
300              instanceStruct inst = SEGinstanceInfo(segSeid)
301              IN (IF inst.gl.sharable
302                  THEN
303                      'SEGuseInfo(child, segd) = use
304                      AND 'SEGinstanceInfo(segSeid) =
305                          STRUCT(inst.gl, inst.refCount + 1, inst.data)
306                  ELSE
307                      (LET seid copy INSET copySet
308                          IN 'TIIinfo(copy) = 'TIIinfo(segSeid)
309                          AND 'SEGinstanceInfo(copy) =
310                              STRUCT(inst.gl, 1, inst.data)
311                          AND 'SEGuseInfo(child, segd) =
312                              STRUCT(copy, use.vloc, use.da)
313                          AND (FORALL segDes segdl ~= segd
314                              : 'SEGuseInfo(child, segdl).instance
315                                  ~= 'SEGuseInfo(child, segd).instance));
316
317
318      END MODULE

```

```

1  $( "      MODULE      smx.specs (version 2.29)
2           CONTENTS      Security Model
3           TYPE:         SPECIAL.specifications
4           LAST CHANGED: 10/12/79. 10:11:23
5  " )
6
7
8  MODULE smx
9
10
11 $( This module now includes what used to be the contents of smx. prv. tif.
12    syl and sen modules)
13
14     TYPES
15
16     $(from smx -- exportable)
17     seid: DESIGNATOR;
18     secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
19                       tExtent, tNull};
20     privType: {
21         privFileUpdateStatus,    privLink,        privLockSeg,
22         privModifyPriv           privMount,
23         privSetFileLevel,        privSetSegProcLevel,
24         privStickySeg,           privTerminalLock,
25         privViolSimpSecurity,     privViolStarSecurity,
26         privViolSimpIntegrity,    privViolStarIntegrity,
27         privViolDiscrAccess,     privSignal,      privWalkPTable,
28         privHalt,                privKernelCall, privViolCompartments,
29         privRealizeExecPermissions};
30
31     daMode: {daRead, daWrite, daExecute};
32     securityCat: DESIGNATOR;
33     integrityCat: DESIGNATOR;
34     domainType: {userDomain, supervisorDomain};
35
36     $(from smx -- redeclarable)
37     nonDisType: STRUCT OF(
38         INTEGER securityLevel; SET OF securityCat securityCatS;
39         INTEGER integrityLevel; SET OF integrityCat integrityCatS);
40     $(integrityCat is typically the null set)
41     daType: SET OF daMode;
42     modeStruct: STRUCT OF(daType ownerMode, groupMode, allMode);
43     tifStruct: STRUCT OF(nonDisType nd;
44         modeStruct da; INTEGER owner, group; SET OF privType priv);
45
46
47     PARAMETERS
48
49     INTEGER SENmaxIndex $(maximum index component of a seid. 2^24 - 1),
50     SENmaxNsp $(maximum nsp component of a seid, 2^8 - 1);
51     INTEGER SENlowLevel, $(system low level)
52     SENhighLevel; $(system high level)
53
54
55     ASSERTIONS
56

```

```

57 FORALL seid s1 s2 (s1 = s2) = (SENseidNsp(s1) = SENseidNsp(s2)
58                               AND SENseidIndex(s1) = SENseidIndex(s2));
59 $(this states that the nsp and index are isomorphic with the seid, and
60 thus uniquely identify it)
61
62 SENlowLevel < SENhighLevel;
63 $(defines the "greater than" relation for security in terms of the integer
64 relation ">")
65
66 SYLgetHigh() INSET {SENlowLevel .. SENhighLevel};
67 $(the current system high level is always within range)
68
69 FORALL seid s | TIIinfo(s) ~= ?
70 : LET nonDisType nd = TIIinfo(s).nd
71   IN nd.securityLevel INSET {SENlowLevel .. SENhighLevel}
72   AND nd.integrityLevel INSET {SENlowLevel .. SENhighLevel};
73 $(restricts the values of the security and integrity levels for any
74 existing objects)
75
76
77 FUNCTIONS
78
79 $(----- sen -- state functions -----)
80
81 VFUN SENseidNsp(seid anySeid) -> INTEGER nsp; $(SENseidNsp)
82 $( this is the nsp table entry component of a seid)
83 INITIALLY
84   nsp INSET {0 .. SENmaxNsp}; $(constrained by assertion above)
85
86 VFUN SENseidIndex(seid anySeid) -> INTEGER index; $(SENseidIndex)
87 $( this is the index component for a seid)
88 INITIALLY
89   index INSET {0 .. SENmaxIndex};
90 $(also characterized by assertion)
91
92 VFUN SENnspType(INTEGER nsp) -> secureEntityType set; $(SENnspType)
93 $( gives the type information as a function of the nsp component of
94 a seid)
95 INITIALLY NOT nsp INSET {0 .. SENmaxNsp} => set = ?;
96
97 $(----- sen -- seid and nsp manipulation -----)
98
99 VFUN SENseidToInt(seid anySeid) -> INTEGER i; $(SENseidToInt)
100 $(gives the integer corresponding to a given seid)
101 DERIVATION
102   SENseidIndex(anySeid) + 2^24 * SENseidNsp(anySeid);
103
104 VFUN SENseidType(seid s) -> secureEntityType set; $(SENseidType)
105 $( returns the type information pertaining to a given seid)
106 DERIVATION
107   LET secureEntityType set1 = SENnspType(SENseidNsp(s))
108   IN IF set1 = ? THEN tNull ELSE set1;
109
110 VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid rSeid;
111 $(SENmakeSeid)
112 $( forms a seid with an nsp the same as the example seid and the given

```

```

113     index; seid allocation is now done by the type managers for the
114     objects in question, allowing seids to be reused in the case of
115     objects that are dynamically allocated)
116     ASSERTIONS
117     index INSET {0 .. SENmaxIndex};
118     DERIVATION
119     SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSeid)
120                 AND SENseidIndex(s) = index;
121
122 $(----- syl functions -----)
123
124 VFUN SYLgetHigh() -> INTEGER level; $(SYLgetHigh)
125 $( represents the current highest security level for the system)
126     HIDDEN;
127     INITIALLY
128     level = SENhighLevel;
129
130 OFUN SYLsetHigh(INTEGER level); $(SYLsetHigh)
131 $( sets the current highest security level for the system to the specified
132    value)
133     EXCEPTIONS
134     KEsylvTooHigh: NOT level INSET {SENlowLevel .. SENhighLevel};
135     EFFECTS
136     'SYLgetHigh() = level;
137
138 $(----- tii state function -----)
139
140 VFUN TIIinfo(seid s) -> tiiStruct st; $(TIIinfo)
141 $(returns the type-independent information for a system object; this
142    information includes discretionary, non-discretionary, and domain access
143    controls, privileges, and the owner and group for the object)
144     HIDDEN;
145     INITIALLY st = ?;
146
147 $(----- smx functions -----)
148
149 VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b; $(SMXhasPriv)
150 $( tells whether a given object -- usually a process -- has a particular
151    privilege)
152     DERIVATION
153     IF TIIinfo(pSeid) ~= ? THEN priv INSET TIIinfo(pSeid).priv ELSE FALSE;
154
155 VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b; $(SMXflow)
156 $( tells whether a given subject "pSeid" can access a given object "oSeid"
157    with the information flow specified by "flow", according to the
158    constraints of the military multilevel security model: these constraints
159    do not apply if the subject has the proper privilege)
160     DEFINITIONS
161     tiiStruct pTii IS TIIinfo(pSeid);
162     tiiStruct oTii IS TIIinfo(oSeid);
163     DERIVATION
164     IF pTii ~= ? AND oTii ~= ?
165     THEN (daWrite INSET da
166          => (NOT SMXhasPriv(pSeid, privViolStarSecurity)
167              => pTii.nd.securityLevel <= oTii.nd.securityLevel)
168          AND (NOT SMXhasPriv(pSeid, privViolCompartments)

```

```

169             => pTii.nd.securityCatS SUBSET oTii.nd.securityCatS)
170         AND (NOT SMXhasPriv(pSeid, privViolSimpIntegrity)
171             => pTii.nd.integrityLevel >= oTii.nd.integrityLevel)
172         AND (NOT SMXhasPriv(pSeid, privViolCompartments)
173             => oTii.nd.integrityCatS SUBSET pTii.nd.integrityCatS))
174     AND (daRead INSET da
175         => (NOT SMXhasPriv(pSeid, privViolSimpSecurity)
176             => oTii.nd.securityLevel <= pTii.nd.securityLevel)
177         AND (NOT SMXhasPriv(pSeid, privViolCompartments)
178             => oTii.nd.securityCatS
179                 SUBSET pTii.nd.securityCatS)
180         AND (NOT SMXhasPriv(pSeid, privViolStarIntegrity)
181             => oTii.nd.integrityLevel
182                 >= pTii.nd.integrityLevel)
183         AND (NOT SMXhasPriv(pSeid, privViolCompartments)
184             => pTii.nd.integrityCatS
185                 SUBSET oTii.nd.integrityCatS))
186     ELSE FALSE:
187
188 VFUN SMXdap(seid pSeid, oSeid: daType da) -> BOOLEAN b:          $(SMXdap)
189 $( tells whether a given subject "pSeid" can access a particular object
190   "oSeid" according to the discretionary access rules of the system --
191   similar to those of UNIX)
192 DEFINITIONS
193   tiiStruct p IS TIIinfo(pSeid);
194   modeStruct mst IS TIIinfo(oSeid).da;
195   tiiStruct o IS TIIinfo(oSeid);
196   BOOLEAN access(daType requested, allowed)
197     IS requested
198       SUBSET allowed
199         UNION (IF SMXhasPriv(pSeid, privRealizeExecPermissions)
200             AND daExecute INSET allowed
201             THEN {daRead}
202             ELSE {});
203 DERIVATION
204   IF o ~= ?
205     THEN SMXhasPriv(pSeid, privViolDiscrAccess)
206         OR access(da, mst.allMode)
207         OR (p.group = o.group AND access(da, mst.groupMode))
208         OR (p.owner = o.owner AND access(da, mst.ownerMode))
209     ELSE FALSE:
210
211 $(----- tii -- extraction and insertion functions-----)
212
213 OFUN TIIcreateEntityLevel(seid oSeid; tiiStruct ntii):$(TIIcreateEntityLevel)
214 $( Initializes the tii information for an object "oSeid": the object must
215   not have currently defined tii information; this is a service function
216   required for the creation of all types of objects in KSOS)
217 ASSERTIONS
218   TIIinfo(oSeid) = ?:
219   ntii.nd.securityLevel INSET {SENlowLevel .. SYLgetHigh()};
220   ntii.nd.integrityLevel INSET {SENlowLevel .. SENhighLevel};
221 EFFECTS
222   'TIIinfo(oSeid) = ntii;
223
224 VFUN TIIgetEntityLevel(seid pSeid, oSeid) -> tiiStruct ntii:

```

```

225                                     $(TIIgetEntityLevel)
226 $(Retrieves the tii information of an object named by "oSeid", as
227   directed by process "pSeid"; mandatory and discretionary checks are also
228   performed; this function is used by
229   functions of the object-maintaining modules, which provide status
230   information to the object that is concerned with getting and setting
231   object levels)
232 EXCEPTIONS
233   KEtiiNoObj: TIIinfo(oSeid) = ? OR NOT SMXflow(pSeid, oSeid, {daRead});
234 DERIVATION
235   TIIinfo(oSeid);
236
237 OFUN TIIsetEntityLevel(seid pSeid, oSeid: tiiStruct ntii):
238                                     $(TIIsetEntityLevel)
239 $( sets the type-independent information for an existing object
240   "oSeid" to the new value "ntii", as directed by process
241   "pSeid", the privilege to set level is always required; if an
242   object's privileges are to be modified, the privilege to modify
243   privileges is required; because only privileged programs are allowed to
244   invoke this function, no other security checks -- either mandatory
245   or discretionary -- are made)
246 DEFINITIONS
247   tiiStruct otii IS TIIinfo(oSeid);
248 EXCEPTIONS
249   $(privilege checking occurs in lev module)
250   KEtiiNoSetPriv:
251     otii.priv ~= ntii.priv AND NOT SMXhasPriv(pSeid, privModifyPriv);
252   KEtiiNoObj: otii = ?;
253 ASSERTIONS
254   ntii.nd.securityLevel INSET {SENlowLevel .. SYLgetHigh()};
255   ntii.nd.integrityLevel INSET {SENlowLevel .. SENhighLevel};
256 EFFECTS
257   'TIIinfo(oSeid) = ntii:
258
259 OFUN TIIclearEntityLevel(seid oSeid);                                     $(TIIclearEntityLevel)
260 $( deletes the type-independent information for an object "oSeid"; this
261   function is used for implementing functions that delete objects -- and
262   thus must also delete the tii info)
263 EFFECTS
264   'TIIinfo(oSeid) = ?;
265
266
267 END MODULE

```

KSOS Kernel Verification Results

5. Appendix B - Sample Code Proof

PROOF OF SMXcompare

;here's the Modula program to be proven:

MODULE SMX;

 DEFINE SMXcompare;
 USE subsetop;

 PROCEDURE SMXcompare(Lt11, Rt11: t11struct): boolean;

 BEGIN

 SMXcompare := (Lt11.nd.securitylevel <= Rt11.nd.securitylevel) AND
 subsetop(Lt11.nd.securitycats, Rt11.nd.securitycats) AND
 (Lt11.nd.integritylevel >= Rt11.nd.integritylevel) AND
 subsetop(Lt11.nd.integritycats, Rt11.nd.integritycats);

 END SMXcompare;

END SMX .

24_MPARSE(SMX.MODULE)
Parsing started
(Parsing Done)
25_PP SMXMODULE

;First, the Modula program is parsed
;with the function MPARSE, in the TRANS.EXE
;environment.
;here's th upper-level spec, entered with
;the Emacs editor, using the Doyer-Moore

```

(SMXCOMPAREMODULE ;Emacs - LISP interface.
  (GVFNS (SMXCOMPARE (LTII HTII)
    (VALUE (AND (NOT (GREATERP (SELECT LTII
      (QUOTE (ND
        SECURITYLEVEL))
        STATE)
      (SELECT HTII
        (QUOTE (ND
          SECURITYLEVEL))
          STATE)))
    (SUBSET (SELECT LTII (QUOTE (ND SECURITYCATS
      **))
      STATE)
    (SELECT HTII (QUOTE (ND SECURITYCATS
      **))
      STATE)
    (NOT (LESSP (SELECT LTII
      (QUOTE (ND
        INTEGRITYLEVEL))
        STATE)
      (SELECT HTII
        (QUOTE (ND
          INTEGRITYLEVEL))
          STATE)))
    (SUBSET (SELECT LTII (QUOTE (ND
      INTEGRITYCATS))
      STATE)
    (SELECT HTII (QUOTE (ND
      INTEGRITYCATS))
      STATE)
    STATE))))))

SMXMODULE
26_PP PRIMITIVEMODULE ;Here's the lower-level spec.

  (PRIMITIVEMODULE (GVFNS (SUBSETOP (X Y)
    (VALUE (SUBSET X Y STATE)))
    (SELECT.RECORD (STRUCTURE FIELD)
      (VALUE (SELECT STRUCTURE FIELD STATE)
        **))
    )
    (GVFNS (SUBSET (X Y))
      (SELECT (STRUCTURE FIELD))))

PRIMITIVEMODULE
27_PP PRS ;This variable PRS has been set to be the
          **
          ;parsed Modula program by MPARSE.

```

```

(MODULE
  Smx
  (DEFINE Smxcompare)
  (USE subsetop)
  (((PROCEDURE
    Smxcompare
    ((CONST (Lt11 Ht11)
      ti1struct))
    NIL
    (NIL
      (BEGIN
        (:= Smxcompare
          (ANDOP (ANDOP (ANDOP (LESSOREQUALOP
            (SELECT.RECORD (SELECT.RECORD
              Lt11
              (QUOTEOP (nd)))
              (QUOTEOP (securitylevel)))
            **
            (SELECT.RECORD (SELECT.RECORD
              Ht11
              (QUOTEOP (nd)))
              (QUOTEOP (securitylevel)))
            **
          )
        (subsetop (SELECT.RECORD
          (SELECT.RECORD
            Lt11
            (QUOTEOP (nd)))
            (QUOTEOP (securitycats)))
          (SELECT.RECORD
            (SELECT.RECORD
              Ht11
              (QUOTEOP (nd)))
              (QUOTEOP (securitycats))))))
        (GREATEROREQUALOP (SELECT.RECORD
          (SELECT.RECORD
            Lt11
            (QUOTEOP (nd)))
            (QUOTEOP (integritylevel)))
          (SELECT.RECORD
            (SELECT.RECORD
              Ht11
              (QUOTEOP (nd)))
              (QUOTEOP (integritylevel))))))
        (subsetop (SELECT.RECORD (SELECT.RECORD Lt11
          (QUOTEOP
            (nd)))
          (QUOTEOP (integritycats)))
          (SELECT.RECORD (SELECT.RECORD Ht11
            (QUOTEOP
              (nd)))
            (QUOTEOP (integritycats))))))
        NIL))))
  NIL))

```

PRS

```

28_ARGLIST(TRANSLATE)      ;Just checking to see the order of the
(PARSED-PROGRAM UPPERMODULE LOWERMODULE)      ;arguments to TRANSLATE.
29_(TRANSLATE PRS SIXMODULE PRIMITIVE)        ;Invoking TRANSLATE on the
;parsed program, the upper-level specs, and
;the lower-level specs.
About to undo DEFINE lists
  DEFINE lists undone
About to disambiguate name duplications
  Renaming done           ;This stuff just messages from TRANSLATE.
About to make integer substitutions for Types and Constants
  Constants and Types replaced by integers
About to perform translation
  Translating finished
About to do some optimization
  Optimizing done
About to splice in exception handling jumps
  Splicing done

```

(Processing Completed^G^G^G^G^G^G^G^G^G^G^G^G^G)

30_PP RUN-PROC ;this is the result of the TRANSLATE call.

((VFMS)

```

(UVFMS (SMXCOMPARE ((ASSIGN S0001 (QUOTEOP (ND)))
(ASSIGN S0002 (SELECT.RECORD LTII S0001))
(ASSIGN S0003 (QUOTEOP (SECURITYLEVEL)))
(ASSIGN S0004 (SELECT.RECORD S0002 S0003))
(ASSIGN S0005 (QUOTEOP (ND)))
(ASSIGN S0006 (SELECT.RECORD HTII S0005))
(ASSIGN S0007 (QUOTEOP (SECURITYLEVEL)))
(ASSIGN S0008 (SELECT.RECORD S0006 S0007))
(ASSIGN S0009 (LESSOREQUALOP S0004 S0008))
(ASSIGN S0010 (QUOTEOP (ND)))
(ASSIGN S0011 (SELECT.RECORD LTII S0010))
(ASSIGN S0012 (QUOTEOP (SECURITYCATS)))
(ASSIGN S0013 (SELECT.RECORD S0011 S0012))
(ASSIGN S0014 (QUOTEOP (ND)))
(ASSIGN S0015 (SELECT.RECORD HTII S0014))
(ASSIGN S0016 (QUOTEOP (SECURITYCATS)))
(ASSIGN S0017 (SELECT.RECORD S0015 S0016))
(ASSIGN S0018 (SUBSETOP S0013 S0017))
(ASSIGN S0019 (ANDOP S0009 S0018))
(ASSIGN S0020 (QUOTEOP (ND)))
(ASSIGN S0021 (SELECT.RECORD LTII S0020))
(ASSIGN S0022 (QUOTEOP (INTEGRITYLEVEL)))
(ASSIGN S0023 (SELECT.RECORD S0021 S0022))
(ASSIGN S0024 (QUOTEOP (ND)))
(ASSIGN S0025 (SELECT.RECORD HTII S0024))
(ASSIGN S0026 (QUOTEOP (INTEGRITYLEVEL)))
(ASSIGN S0027 (SELECT.RECORD S0025 S0026))
(ASSIGN S0028 (GREATEROREQUALOP S0023 S0027))

```

```

(ASSIGN S0029 (ANDOP S0019 S0028))
(ASSIGN S0030 (QUOTEOP (ND)))
(ASSIGN S0031 (SELECT.RECORD LTII S0030))
(ASSIGN S0032 (QUOTEOP (INTEGRITYCATS)))
(ASSIGN S0033 (SELECT.RECORD S0031 S0032))
(ASSIGN S0034 (QUOTEOP (ND)))
(ASSIGN S0035 (SELECT.RECORD HTII S0034))
(ASSIGN S0036 (QUOTEOP (INTEGRITYCATS)))
(ASSIGN S0037 (SELECT.RECORD S0035 S0036))
(ASSIGN S0038 (SUBSETOP S0033 S0037))
(ASSIGN AC1 (ANDOP S0029 S0038))
(ANSWER)))

```

```

(SMX NIL))
(INITIALIZATION)
(INVARIANT)
(DEFINITIONS)
(GLOBAL.VARIABLES))

```

```

RUN-PROC
31_DRIBBLE()

```

```

;I saved the specs and translation on a file
;for use in VC generation and proving. Recall
;that the tools for these processes are in a
;different environment: CIFVCG.EXE.

```

```

;So, now I'm in the new environment. I've
;loaded the things I saved previously into
32_(MAKE.VCS RUN-PROC SMXMODULE PRIMITIVEMODULE) ;this environment. I begin
;by invoking MAKE.VCS on the implementation,
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM) ;and the specs.

```

```

;All of the messages appearing here are warnings from the VCG. Looking
;at the CIF (RUN-PROC) you should be able to tell why they're here. Can
;you? (Check back to the Boyer-Moore formalization of HDM.)

```

```

collecting lists
4022, 10166 free cells
(Argument (SECURITYLEVEL) in (QUOTEOP (SECURITYLEVEL)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (SECURITYLEVEL) in (QUOTEOP (SECURITYLEVEL)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (SECURITYCATS) in (QUOTEOP (SECURITYCATS)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (SECURITYCATS) in (QUOTEOP (SECURITYCATS)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (INTEGRITYLEVEL) in (QUOTEOP (INTEGRITYLEVEL)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (INTEGRITYLEVEL) in (QUOTEOP (INTEGRITYLEVEL)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (INTEGRITYCATS) in (QUOTEOP (INTEGRITYCATS)) not a LITATOM)
(Argument (ND) in (QUOTEOP (ND)) not a LITATOM)
(Argument (INTEGRITYCATS) in (QUOTEOP (INTEGRITYCATS)) not a LITATOM)

```


(COMMENT INPUT.TO.OUTPUT)

(PROVE.LEMMA

CONCLUSION NIL

```

(EQUAL (AND (AND (AND (NOT (GREATERP (SELECT (SELECT (LTII*)
                                         (QUOTE (ND))
                                         (STATE*))
                                         (QUOTE (SECURITYLEVEL))
                                         (STATE*))
                                         (SELECT (SELECT (HTII*)
                                         (QUOTE (ND))
                                         (STATE*))
                                         (QUOTE (SECURITYLEVEL))
                                         (STATE*))))))
(SUBSET (SELECT (SELECT (LTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (SECURITYCATS))
          (STATE*))
        (SELECT (SELECT (HTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (SECURITYCATS))
          (STATE*))
        (STATE*)))
(NOT (LESSP (SELECT (SELECT (LTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (INTEGRITYLEVEL))
          (STATE*))
        (SELECT (SELECT (HTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (INTEGRITYLEVEL))
          (STATE*))))))
(SUBSET (SELECT (SELECT (LTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (INTEGRITYCATS))
          (STATE*))
        (SELECT (SELECT (HTII*)
                      (QUOTE (ND))
                      (STATE*))
          (QUOTE (INTEGRITYCATS))
          (STATE*))
        (STATE*)))

```

```

(AND (NOT (GREATERP (SELECT (LTII*)
                             (QUOTE (ND SECURITYLEVEL))
                             (STATE*)))
      (SELECT (HTII*)
              (QUOTE (ND SECURITYLEVEL))
              (STATE*))))))
(SUBSET (SELECT (LTII*)
               (QUOTE (ND SECURITYCATS))
               (STATE*)))
(SELECT (HTII*)
      (QUOTE (ND SECURITYCATS))
      (STATE*)))
(STATE*)))
(NOT (LESSP (SELECT (LTII*)
                   (QUOTE (ND INTEGRITYLEVEL))
                   (STATE*)))
      (SELECT (HTII*)
              (QUOTE (ND INTEGRITYLEVEL))
              (STATE*))))))
(SUBSET (SELECT (LTII*)
               (QUOTE (ND INTEGRITYCATS))
               (STATE*)))
(SELECT (HTII*)
      (QUOTE (ND INTEGRITYCATS))
      (STATE*)))
(STATE*))))))
(UNDO.BACK.THROUGH INPUT.TO.OUTPUT)
(COMMENT STATE.EQUIVALENCE)
(ADD.AXIOM HYPOTHESIS (REWRITE)
  (AND (EQUAL (SUBSET X Y (NEWSTATE))
              (SUBSET X Y (STATE))))
  (EQUAL (SELECT STRUCTURE FIELD (NEWSTATE))
        (SELECT STRUCTURE FIELD (STATE)))))
(PROVE.LEMMA CONCLUSION NIL (AND (EQUAL (SUBSET V1 V2 (NEWSTATE))
                                         (SUBSET V1 V2 (STATE))))
                              (EQUAL (SELECT V1 V2 (NEWSTATE))
                                      (SELECT V1 V2 (STATE)))))
(UNDO.BACK.THROUGH STATE.EQUIVALENCE)
(UNDO.BACK.THROUGH CORRECTNESS.OF.SMXCOMPARE)
(UNDO.BACK.THROUGH
  CORRECTNESS.OF.THE.IMPLEMENTATION.OF.SMXCOMPAREMODULE.ON.PRIMITIVEMODULE))
VCG.RESULT
34_PP AXIOM1 AXIOM2 ;Part of the axiomatization of structures
                    ;requires the following two axioms. They are
(ADD.AXIOM IDENTITY.OF.SELECT (REWRITE) ;added to the list of theorem
  (EQUAL (SELECT S NIL STATE) ;prover events.
    S))

```

```
(ADD.AXIOM ASSOCIATIVITY.OF.SELECT (REWRITE)
  (EQUAL (SELECT (SELECT S P1 STATE)
                P2 STATE)
    (SELECT S (APPEND P1 P2)
      STATE))
  NIL)
(AXIOM1 AXIOM2)
35_DRIBBLE()
```

;Here is the proof of SMX4000ULA on PRIMITIVE MODULE

<LISP>LISP.EXE.132

<MOORE>CODE..4

<HIER>CODE1..2

<MOORE>DATA..4

<HIER>DATA1..2

Friday, November 16, 1979 8:25PM-PST

```
_COMMENT(
CORRECTNESS.OF.THE.IMPLEMENTATION.OF.SMXCOMPAREMODULE.ON.PRIMITIVE MODULE
)
CORRECTNESS.OF.THE.IMPLEMENTATION.OF.SMXCOMPAREMODULE.ON.PRIMITIVE MODULE
```

```
_DCL(STATE NIL)
STATE
```

```
_DCL(STATE* NIL)
STATE*
```

```
_DCL(BEGIN NIL)
BEGIN
```

```
_DCL(NEWSTATE NIL)
NEWSTATE
```

_DCL(NEXT (STATE))
NEXT

_JCL(SUBSET (X Y STATE))
SUBSET

_DCL(SELECT (STRUCTURE FIELD STATE))
SELECT

_DCL(TRUEOP (STATE))
TRUEOP

_DCL(FALSEOP (STATE))
FALSEOP

_JCL(UNDEFOP (STATE))
UNDEFOP

_DCL(EQUALOP (X Y STATE))
EQUALOP

_DCL(NEQUALOP (X Y STATE))
NEQUALOP

_DCL(ZERUPOP (X STATE))
ZERUPOP

_JCL(GREATERPOP (X Y STATE))
GREATERPOP

_DCL(LESSPOP (X Y STATE))
LESSPOP

_DCL(ADD1UP (X STATE))
ADD1UP

_DCL(PLUSOP (X Y STATE))
PLUSOP

```
_DCL(DIFFERENCEOF (X Y STATE))
DIFFERENCEOF
```

```
_DCL(NUMBERPOP (X STATE))
NUMBERPOP
```

```
_DCL(GREATEROREQUALOP (X Y STATE))
GREATEROREQUALOP
```

_JUL(LESSOREQUALOP (Y Y STATE))
LESSUREQUALOP

```
_UCL(URUP (X Y STATE))
URUP
```

_DCL(ANDUP (Y Y STATE))
ANDUP

```
_DCL(QUOTE OF (X STATE))
QUOTECP
```

```

      _DCL(SUBSETOP (X Y STATE))
      SUBSETOP

```

```
_DCL(SELECT.RECORD (STRUCTURE FIELD STATE))
SELECT.RECORD
```

```

_DCL(SMXCOMPARE (LTII FTII STATE))
SMXCOMPARE

```

```

_ADD.AXID(X(IDENTITY.OF.SELECT
                                (REWRITE)
                                (EQUAL (SELECT 3 NIL STATE) S))
IDENTITY.OF.SELECT

```

```

-ADD.AXIOM(ASSOCIATIVITY.OF.SELECT
          (REWRITE)
          (EQUAL (SELECT (SELECT S P1 STATE) P2 STATE)
                 (SELECT S (APPEND P1 P2) STATE)))
          NIL)
ASSOCIATIVITY.OF.SELECT

```

_COMMENT(CORRECTNESS.OF.INITIALIZATION.OF.SMXCOMPAREMODULE)
CORRECTNESS.OF.INITIALIZATION.OF.SMXCOMPAREMODULE

_COMMENT(INPUT.TO.OUTPUT)
INPUT.TO.OUTPUT

_PROVE.LEMMA(CONCLUSION NIL T)
This conjecture simplifies, clearly, to:

(TRUE).

Q.E.D.

Load average during proof: 1.134404
Elapsed time: 1.065 seconds
CPU time (devoted to theorem proving): .124 seconds
GC time: 0.0 seconds
IO time: .006 seconds
CUNses consumed: 50

PROVED

_UNDO.BACK.THROUGH(INPUT.TO.OUTPUT)
((PROVE.LEMMA CONCLUSION NIL (TRUE)) (COMMENT
INPUT.TO.OUTPUT))

_COMMENT(STATE.EQUIVALENCE)
STATE.EQUIVALENCE

_ADD.AXIOM(HYPOTHESIS
 (REWRITE)
 (AND (EQUAL (SUBSET X Y (NEWSTATE))
 (SUBSET X Y (STATE)))
 (EQUAL (SELECT STRUCTURE FIELD (NEWSTATE))
 (SELECT STRUCTURE FIELD (STATE)))))
HYPOTHESIS

_PROVE.LEMMA(CONCLUSION NIL T)
This formula simplifies, clearly, to:

(TRUE).

Q.E.D.

Load average during proof: 1.202444
Elapsed time: .225 seconds
CPU time (devoted to theorem proving): .107 seconds
GC time: 0.0 seconds
IJ time: .001 seconds
CONSes consumed: 50

PROVED

_UNDO.BACK.THROUGH(STATE.EQUIVALENCE)
((PROVE.LEMMA CONCLUSION NIL (TRUE)) (ADD.AXIOM HYPOTHESIS (REWRITE) (AND (EQUAL (SUBSET X Y (NEWSTATE)) (SUBSET X Y (STATE))) (EQUAL (SELECT STRUCTURE FIELD (NEWSTATE)) (SELECT STRUCTURE FIELD (STATE))))) (COMMENT STATE.EQUIVALENCE))

_UNDO.BACK.THROUGH(
CORRECTNESS.OF.INITIALIZATION.OF.SMXCOMPAREMODULE)
((COMMENT CORRECTNESS.OF.INITIALIZATION.OF.SMXCOMPAREMODULE)
)

_COMMENT(CORRECTNESS.OF.SMXCOMPARE)
CORRECTNESS.OF.SMXCOMPARE

_OCL(HT11* NIL)
HT11*

_OCL(LT11* NIL)
LT11*

_COMMENT(INPUT.TO.OUTPUT)
INPUT.TO.OUTPUT

AD-A111 563

FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 17/2
(KSOS) KERNEL VERIFICATION RESULTS. KERNELIZED SECURE OPERATING--ETC(U)
DEC 80

UNCLASSIFIED WDL-TR9001

NL

2 of 2

31 MAR 81



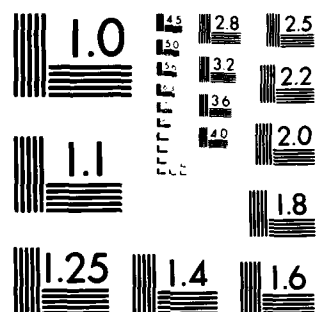
END

DATE

FILED

3-82

DTIC



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

```

( _PROVE.LEMMA(CONCLUSION NIL
  (EQUAL
    (AND
      (AND
        (AND
          (NOT
            (GREATERP
              (SELECT
                (SELECT (LTII*) (QUOTE (ND)) (STATE*))
                (QUOTE (SECURITYLEVEL))
                (STATE*))
              (SELECT
                (SELECT (HTII*) (QUOTE (ND)) (STATE*))
                (QUOTE (SECURITYLEVEL))
                (STATE*))))))
          (SURSET
            (SELECT
              (SELECT (LTII*) (QUOTE (ND)) (STATE*))
              (QUOTE (SECURITYCATS))
              (STATE*))
            (SELECT
              (SELECT (HTII*) (QUOTE (ND)) (STATE*))
              (QUOTE (SECURITYCATS))
              (STATE*))
            (STATE*))))
          (NOT
            (LESSP
              (SELECT
                (SELECT (LTII*) (QUOTE (ND)) (STATE*))
                (QUOTE (INTEGRITYLEVEL))
                (STATE*))
              (SELECT
                (SELECT (HTII*) (QUOTE (ND)) (STATE*))
                (QUOTE (INTEGRITYLEVEL))
                (STATE*))))))
          (SURSET
            (SELECT
              (SELECT (LTII*) (QUOTE (ND)) (STATE*))
              (QUOTE (INTEGRITYCATS))
              (STATE*))
            (SELECT
              (SELECT (HTII*) (QUOTE (ND)) (STATE*))
              (QUOTE (INTEGRITYCATS))
              (STATE*))
            (STATE*))))
          (AND
            (NOT
              (GREATERP (SELECT (LTII*)
                (QUOTE (ND SECURITYLEVEL))
                (STATE*))
                (SELECT (HTII*)
                (QUOTE (ND SECURITYLEVEL))
                (STATE*)))))

```

```

(SUBSET (SELECT (LTII*)
              (QUOTE (ND SECURITYCATS))
              (STATE*)))
  (SELECT (HTII*)
            (QUOTE (ND SECURITYCATS))
            (STATE*)))
  (STATE*)))
( NOT
  (LESSP (SELECT (LTII*)
                (QUOTE (ND INTEGRITYLEVEL))
                (STATE*)))
    (SELECT (HTII*)
              (QUOTE (ND INTEGRITYLEVEL))
              (STATE*))))))
(SUPSET (SELECT (LTII*)
              (QUOTE (ND INTEGRITYCATS))
              (STATE*)))
  (SELECT (HTII*)
            (QUOTE (ND INTEGRITYCATS))
            (STATE*)))
  (STATE*))))))

```

This simplifies, expanding the definitions of GREATERP, NOT, and AND, to 13 new goals:

Case 1. (IMPLIES

```

  (AND
    (LESSP (SELECT (SELECT (HTII*)
                        (CONS (QUOTE ND) NIL)
                        (STATE*)))
              (CONS (QUOTE SECURITYLEVEL) NIL)
              (STATE*)))
      (SELECT (SELECT (LTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
              (CONS (QUOTE SECURITYLEVEL) NIL)
              (STATE*))))
    (LESSP (SELECT (SELECT (LTII*)
                        (CONS (QUOTE ND) NIL)
                        (STATE*)))
              (CONS (QUOTE INTEGRITYLEVEL) NIL)
              (STATE*)))
      (SELECT (SELECT (HTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
              (CONS (QUOTE INTEGRITYLEVEL) NIL)
              (STATE*))))))
  (NOT
    (LESSP
      (SELECT (HTII*)
                (CONS (QUOTE ND)
                    (CONS (QUOTE SECURITYLEVEL) NIL))
                (STATE*)))
    )
  )

```

```

      (SELECT (LTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE SECURITYLEVEL) NIL))
        (STATE*)))
(NOT
 (LESSP
  (SELECT
    (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYLEVEL) NIL))
    (STATE*))
  (SELECT
    (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYLEVEL) NIL))
    (STATE*)))
 (SUBSET
  (SELECT (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYCATS) NIL))
    (STATE*))
  (SELECT (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYCATS) NIL))
    (STATE*)))
 (NOT
  (SUBSET
   (SELECT (LTII*)
     (CONS (QUOTE ND)
       (CONS (QUOTE SECURITYCATS) NIL))
     (STATE*))
   (SELECT (HTII*)
     (CONS (QUOTE ND)
       (CONS (QUOTE SECURITYCATS) NIL))
     (STATE*))
   (STATE*))))).

```

This again simplifies, rewriting with
 ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
 expanding the function APPEND, to:

(TRUE).

Case 2. (IMPLIES

(AND

(LESSP (SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*))
(SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*)))))

(NOT

(LESSP (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*))
(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*)))))

(NOT

(LESSP
(SELECT (HTII*)
(CONS (QUOTE ND)
(CONS (QUOTE SECURITYLEVEL) NIL))
(STATE*))
(SELECT (LTII*)
(CONS (QUOTE ND)
(CONS (QUOTE SECURITYLEVEL) NIL))
(STATE*)))))

(NOT

(LESSP
(SELECT
(LTII*)
(CONS (QUOTE ND)
(CONS (QUOTE INTEGRITYLEVEL) NIL))
(STATE*))
(SELECT
(HTII*)
(CONS (QUOTE ND)
(CONS (QUOTE INTEGRITYLEVEL) NIL))
(STATE*)))))

(SUBSET

(SELECT (LTII*)
(CONS (QUOTE ND)
(CONS (QUOTE INTEGRITYCATS) NIL))
(STATE*))

```

      (SELECT (HTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (STATE*)))
(NOT
 (SURSET
  (SELECT (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYCATS) NIL))
    (STATE*))
  (SELECT (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYCATS) NIL))
    (STATE*))
  (STATE*))))),

```

which we again simplify, applying the lemmas
ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
unfolding the function APPEND, to:

(TRUE).

Case 3. (IMPLIES

```

  (AND
   (NOT
    (LESSP (SELECT (SELECT (HTII*)
      (CONS (QUOTE ND) NIL)
      (STATE*))
      (CONS (QUOTE SECURITYLEVEL) NIL)
      (STATE*))
      (SELECT (SELECT (LTII*)
        (CONS (QUOTE ND) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYLEVEL) NIL)
        (STATE*))))
    (SURSET (SELECT (SELECT (LTII*)
      (CONS (QUOTE ND) NIL)
      (STATE*))
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*))
      (SELECT (SELECT (HTII*)
        (CONS (QUOTE ND) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYCATS) NIL)
        (STATE*))
      (STATE*)))
    (STATE*)))
  )

```

```

(LESSP (SELECT (SELECT (LTII*)
                      (CONS (QUOTE ND) NIL)
                      (STATE*)))
      (CONS (QUOTE INTEGRITYLEVEL) NIL)
      (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE ND) NIL)
          (STATE*)))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*)))
(NOT
(LESSP
 (SELECT (HTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE SECURITYLEVEL) NIL))
          (STATE*)))
 (SELECT (LTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE SECURITYLEVEL) NIL))
          (STATE*))))))
(NOT
(LESSP
 (SELECT
  (LTII*)
  (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
  (STATE*)))
 (SELECT
  (HTII*)
  (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
  (STATE*))))))
(SUBSET
 (SELECT (LTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE INTEGRITYCATS) NIL))
          (STATE*)))
 (SELECT (HTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE INTEGRITYCATS) NIL))
          (STATE*)))
 (STATE*)))
(NOT
 (SUBSET
  (SELECT (LTII*)
           (CONS (QUOTE ND)
                 (CONS (QUOTE SECURITYCATS) NIL))
           (STATE*)))
  (SELECT (HTII*)
           (CONS (QUOTE ND)
                 (CONS (QUOTE SECURITYCATS) NIL))
           (STATE*)))
  (STATE*))))).

```

But this simplifies again, applying
ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
expanding the function APPEND, to:

(TRUE).

Case 4. (IMPLIES

(AND

(NOT

(LESSP (SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*))

(SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*)))))

(SUBSET (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYCATS) NIL)
(STATE*))

(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYCATS) NIL)
(STATE*))

(STATE*))

(NOT

(LESSP (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*))

(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*)))))

(SUBSET (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYCATS) NIL)
(STATE*))

(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYCATS) NIL)
(STATE*))

(STATE*))

```

(LESSP
  (SELECT (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*)))
  (SELECT (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*)))
(NOT
  (LESSP
    (SELECT (LTII*)
      (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
      (STATE*)))
    (SELECT (HTII*)
      (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
      (STATE*))))).

```

But this again simplifies, rewriting with
 ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
 expanding the function APPEND, to:

(TRUE).

Case 5. (IMPLIES

(AND

(NOT

```

  (LESSP (SELECT (SELECT (HTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE SECURITYLEVEL) NIL)
    (STATE*)))
    (SELECT (SELECT (LTII*)
      (CONS (QUOTE ND) NIL)
      (STATE*))
      (CONS (QUOTE SECURITYLEVEL) NIL)
      (STATE*))))
  (SUBSET (SELECT (SELECT (LTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE SECURITYCATS) NIL)
    (STATE*))
    (SELECT (SELECT (HTII*)
      (CONS (QUOTE ND) NIL)
      (STATE*))
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*))
    (STATE*)))

```

```

(NEG
  (LESSP (SELECT (SELECT (LTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE INTEGRITYLEVEL) NIL)
    (STATE*))
    (SELECT (SELECT (HTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE INTEGRITYLEVEL) NIL)
    (STATE*))))
  (SUBSET (SELECT (SELECT (LTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE INTEGRITYCATS) NIL)
    (STATE*))
    (SELECT (SELECT (HTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*))
    (CONS (QUOTE INTEGRITYCATS) NIL)
    (STATE*))
    (STATE*)))
  (LESSP
    (SELECT (HTII*)
    (CONS (QUOTE ND)
    (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*))
    (SELECT (LTII*)
    (CONS (QUOTE ND)
    (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*)))
  (NOT
    (LESSP
      (SELECT
        (LTII*)
        (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
        (STATE*))
      (SELECT
        (HTII*)
        (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
        (STATE*)))
    (SUBSET
      (SELECT (LTII*)
        (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (SELECT (HTII*)
        (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (STATE*)))

```

```

(NOT
 (SUBSET
  (SELECT (LTII*)
   (CONS (QUOTE ND)
    (CONS (QUOTE SECURITYCATS) NIL))
   (STATE*)))
 (SELECT (HTII*)
  (CONS (QUOTE ND)
   (CONS (QUOTE SECURITYCATS) NIL))
  (STATE*)))
 (STATE*))))),

```

which we again simplify, applying ASSOCIATIVITY.OF.SELECT,
CAR.CONS, and CDR.CONS, and expanding the function APPEND,
to:

(TRUE).

Case 6. (IMPLIES

```

 (AND
  (NOT
   (LESSP (SELECT (SELECT (HTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*)))
    (CONS (QUOTE SECURITYLEVEL) NIL)
    (STATE*)))
   (SELECT (SELECT (LTII*)
    (CONS (QUOTE ND) NIL)
    (STATE*)))
    (CONS (QUOTE SECURITYLEVEL) NIL)
    (STATE*))))
 (SUBSET (SELECT (SELECT (LTII*)
  (CONS (QUOTE ND) NIL)
  (STATE*)))
  (CONS (QUOTE SECURITYCATS) NIL)
  (STATE*)))
 (SELECT (SELECT (HTII*)
  (CONS (QUOTE ND) NIL)
  (STATE*)))
  (CONS (QUOTE SECURITYCATS) NIL)
  (STATE*)))
 (STATE*)))
 (NOT
  (LESSP (SELECT (SELECT (LTII*)
   (CONS (QUOTE ND) NIL)
   (STATE*)))
   (CONS (QUOTE INTEGRITYLEVEL) NIL)
   (STATE*)))
  (SELECT (SELECT (HTII*)
   (CONS (QUOTE ND) NIL)
   (STATE*)))
   (CONS (QUOTE INTEGRITYLEVEL) NIL)
   (STATE*))))))

```

```

(SUBSET (SELECT (SELECT (LTII*)
                        (CONS (QUOTE NO) NIL)
                        (STATE*)))
        (CONS (QUOTE INTEG'ITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
            (CONS (QUOTE NO) NIL)
            (STATE*)))
        (CONS (QUOTE INTEGRITYCATS) NIL)
        (STATE*)))
(LESSP
 (SELECT (HTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE SECURITYLEVEL) NIL))
         (STATE*)))
 (SELECT (LTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE SECURITYLEVEL) NIL))
         (STATE*)))
(NCT
 (LESSP
  (SELECT
   (LTII*)
   (CONS (QUOTE NO)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
   (STATE*)))
  (SELECT
   (HTII*)
   (CONS (QUOTE NO)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
   (STATE*))))
(SUBSET
 (SELECT (LTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE INTEGRITYCATS) NIL))
         (STATE*)))
 (SELECT (HTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE INTEGRITYCATS) NIL))
         (STATE*)))
 (STATE*)))
(SUBSET
 (SELECT (LTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE SECURITYCATS) NIL))
         (STATE*)))
 (SELECT (HTII*)
         (CONS (QUOTE NO)
              (CONS (QUOTE SECURITYCATS) NIL))
         (STATE*)))
 (STATE*)))

```

which again simplifies, applying ASSOCIATIVITY.OF.SELECT,
CAR.CONS, and CDR.CONS, and opening up the definition of
APPEND, to:

(TRUE).

Case 7. (IMPLIES
(AND
(NOT

```

      (LESSP (SELECT (SELECT (HTII*)
                            (CONS (QUOTE ND) NIL)
                            (STATE*)))
             (CONS (QUOTE SECURITYLEVEL) NIL)
             (STATE*)))
      (SELECT (SELECT (LTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
      (CONS (QUOTE SECURITYLEVEL) NIL)
      (STATE*))))
(SUBSET (SELECT (SELECT (LTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
        (CONS (QUOTE SECURITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
            (CONS (QUOTE ND) NIL)
            (STATE*)))
(CONS (QUOTE SECURITYCATS) NIL)
(STATE*))
(NOT
 (LESSP (SELECT (SELECT (LTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
        (CONS (QUOTE INTEGRITYLEVEL) NIL)
        (STATE*)))
 (SELECT (SELECT (HTII*)
                (CONS (QUOTE ND) NIL)
                (STATE*)))
        (CONS (QUOTE INTEGRITYLEVEL) NIL)
        (STATE*))))
(SUBSET (SELECT (SELECT (LTII*)
                    (CONS (QUOTE ND) NIL)
                    (STATE*)))
        (CONS (QUOTE INTEGRITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
            (CONS (QUOTE ND) NIL)
            (STATE*)))
(CONS (QUOTE INTEGRITYCATS) NIL)
(STATE*))
(STATE*))

```

```

(LESSP
  (SELECT (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*)))
  (SELECT (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE SECURITYLEVEL) NIL))
    (STATE*)))
(NOT
  (LESSP
    (SELECT
      (LTII*)
      (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
      (STATE*)))
    (SELECT
      (HTII*)
      (CONS (QUOTE ND)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
      (STATE*))))))
(SURSET
  (SELECT (LTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYCATS) NIL))
    (STATE*)))
  (SELECT (HTII*)
    (CONS (QUOTE ND)
      (CONS (QUOTE INTEGRITYCATS) NIL))
    (STATE*)))
  (STATE*)))

```

which again simplifies, applying ASSOCIATIVITY.OF.SELECT,
 CAR.CONS, and CDR.CONS, and unfolding APPEND, to:

(TRUE).

Case 3. (IMPLIES

(AND

(NOT

(LESSP (SELECT (SELECT (HTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE SECURITYLEVEL) NIL)

(STATE*))

(SELECT (SELECT (LTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE SECURITYLEVEL) NIL)

(STATE*))))

(SUBSET (SELECT (SELECT (LTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE SECURITYCATS) NIL)

(STATE*))

(SELECT (SELECT (HTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE SECURITYCATS) NIL)

(STATE*))

(STATE*))

(NOT

(LESSP (SELECT (SELECT (LTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYLEVEL) NIL)

(STATE*))

(SELECT (SELECT (HTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYLEVEL) NIL)

(STATE*))))

(SUBSET (SELECT (SELECT (LTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYCATS) NIL)

(STATE*))

(SELECT (SELECT (HTII*)

(CONS (QUOTE NO) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYCATS) NIL)

(STATE*))

(STATE*))

(NOT

(LESSP

(SELECT (HTII*)

(CONS (QUOTE NO)

(CONS (QUOTE SECURITYLEVEL) NIL))

(STATE*))

```

      (SELECT (LTII*)
        (CONS (QUOTE NO)
          (CONS (QUOTE SECURITYLEVEL) NIL))
        (STATE*))))
    (NOT
      (LESSP
        (SELECT (ITII*)
          (CONS (QUOTE NO)
            (CONS (QUOTE INTEGRITYLEVEL) NIL))
          (STATE*)))
        (SELECT (HTII*)
          (CONS (QUOTE NO)
            (CONS (QUOTE INTEGRITYLEVEL) NIL))
          (STATE*))))).

```

However this again simplifies, rewriting with
 ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
 opening up APPEND, to:

(TRUE).

Case 9. (IMPLIES

```

  (AND
    (NOT
      (LESSP (SELECT (SELECT (HTII*)
        (CONS (QUOTE NO) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYLEVEL) NIL)
        (STATE*))
        (SELECT (SELECT (LTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*))
          (CONS (QUOTE SECURITYLEVEL) NIL)
          (STATE*))))
      (SUBSET (SELECT (SELECT (LTII*)
        (CONS (QUOTE NO) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYCATS) NIL)
        (STATE*))
        (SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*))
          (CONS (QUOTE SECURITYCATS) NIL)
          (STATE*))
          (STATE*)))
      (NOT

```

```

(LESSP (SELECT (SELECT (LTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
      (CONS (QUOTE INTEGRITYLEVEL) NIL)
      (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*)))
(SUBSET (SELECT (SELECT (LTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
      (CONS (QUOTE INTEGRITYCATS) NIL)
      (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
(CONS (QUOTE INTEGRITYCATS) NIL)
(STATE*)))
(STATE*)))
(NOT
(LESSP
(SELECT (HTII*)
      (CONS (QUOTE NO)
            (CONS (QUOTE SECURITYLEVEL) NIL))
      (STATE*)))
(SELECT (LTII*)
      (CONS (QUOTE NO)
            (CONS (QUOTE SECURITYLEVEL) NIL))
      (STATE*))))
(NOT
(LESSP
(SELECT
  (LTII*)
  (CONS (QUOTE NO)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
  (STATE*)))
(SELECT
  (HTII*)
  (CONS (QUOTE NO)
        (CONS (QUOTE INTEGRITYLEVEL) NIL))
  (STATE*))))
(SUBSET
(SELECT (LTII*)
      (CONS (QUOTE NO)
            (CONS (QUOTE INTEGRITYCATS) NIL))
      (STATE*)))
(SELECT (HTII*)
      (CONS (QUOTE NO)
            (CONS (QUOTE INTEGRITYCATS) NIL))
      (STATE*)))
(STATE*)))

```

```

(SUBSET
  (SELECT (LTII*)
    (CONS (QUOTE NO)
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*)))
  (SELECT (HTII*)
    (CONS (QUOTE NO)
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*)))
  (STATE*))),

```

which again simplifies, rewriting with
ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
opening up APPEND, to:

(TRUE).

Case 19.(IMPLIES

```

  (AND
    (NOT
      (LESSP (SELECT (SELECT (HTII*)
        (CONS (QUOTE NO) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYLEVEL) NIL)
        (STATE*))
        (SELECT (SELECT (LTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*))
          (CONS (QUOTE SECURITYLEVEL) NIL)
          (STATE*))))))
    (SUBSET (SELECT (SELECT (LTII*)
      (CONS (QUOTE NO) NIL)
      (STATE*))
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*))
      (SELECT (SELECT (HTII*)
        (CONS (QUOTE NO) NIL)
        (STATE*))
        (CONS (QUOTE SECURITYCATS) NIL)
        (STATE*))
      (STATE*)))
    (NOT
      (LESSP (SELECT (SELECT (LTII*)
        (CONS (QUOTE NO) NIL)
        (STATE*))
        (CONS (QUOTE INTEGRITYLEVEL) NIL)
        (STATE*))
        (SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*))
          (CONS (QUOTE INTEGRITYLEVEL) NIL)
          (STATE*))))))
  )

```

```

(SUBSET (SELECT (SELECT (LTII*)
                        (CONS (QUOTE ND) NIL)
                        (STATE*)))
        (CONS (QUOTE INTEGRITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
            (CONS (QUOTE ND) NIL)
            (STATE*)))
(CONS (QUOTE INTEGRITYCATS) NIL)
(STATE*)))
(NOT
 (LESSP
  (SELECT (HTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE SECURITYLEVEL) NIL))
          (STATE*)))
  (SELECT (LTII*)
          (CONS (QUOTE ND)
                (CONS (QUOTE SECURITYLEVEL) NIL))
          (STATE*))))))
(NOT
 (LESSP
  (SELECT
   (LTII*)
   (CONS (QUOTE ND)
         (CONS (QUOTE INTEGRITYLEVEL) NIL))
   (STATE*)))
  (SELECT
   (HTII*)
   (CONS (QUOTE ND)
         (CONS (QUOTE INTEGRITYLEVEL) NIL))
   (STATE*))))))
(SUBSET
 (SELECT (LTII*)
         (CONS (QUOTE ND)
               (CONS (QUOTE INTEGRITYCATS) NIL))
         (STATE*)))
 (SELECT (HTII*)
         (CONS (QUOTE ND)
               (CONS (QUOTE INTEGRITYCATS) NIL))
         (STATE*)))
 (STATE*))).

```

But this again simplifies, rewriting with the lemmas ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and opening up APPEND, to:

```
(TRUE).
```

Case 11. (IMPLIES

(AND

(NOT

```

(LESSP (SELECT (SELECT (HTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
      (CONS (QUOTE SECURITYLEVEL) NIL)
      (STATE*)))
(SELECT (SELECT (LTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
      (CONS (QUOTE SECURITYLEVEL) NIL)
      (STATE*)))
(SUPSET (SELECT (SELECT (LTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
        (CONS (QUOTE SECURITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
      (CONS (QUOTE SECURITYCATS) NIL)
      (STATE*)))
(STATE*)))

```

(NOT

```

(LESSP (SELECT (SELECT (LTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
      (CONS (QUOTE INTEGRITYLEVEL) NIL)
      (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
      (CONS (QUOTE INTEGRITYLEVEL) NIL)
      (STATE*)))
(STATE*)))

```

(NOT

```

(SUPSET (SELECT (SELECT (LTII*)
                      (CONS (QUOTE NO) NIL)
                      (STATE*)))
        (CONS (QUOTE INTEGRITYCATS) NIL)
        (STATE*)))
(SELECT (SELECT (HTII*)
          (CONS (QUOTE NO) NIL)
          (STATE*)))
      (CONS (QUOTE INTEGRITYCATS) NIL)
      (STATE*)))
(STATE*)))

```

(NOT

(LESSP

```

      (SELECT (HTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE SECURITYLEVEL) NIL))
        (STATE*))
      (SELECT (LTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE SECURITYLEVEL) NIL))
        (STATE*)))
    (NOT
      (LESSP
        (SELECT
          (LTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE INTEGRITYLEVEL) NIL))
          (STATE*))
        (SELECT
          (HTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE INTEGRITYLEVEL) NIL))
          (STATE*))))
    (SUBSET
      (SELECT (LTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (SELECT (HTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (STATE*)))
    (NOT
      (SUBSET
        (SELECT (LTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE SECURITYCATS) NIL))
          (STATE*))
        (SELECT (HTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE SECURITYCATS) NIL))
          (STATE*))
        (STATE*))))),

```

which we again simplify, applying ASSOCIATIVITY.OF.SELECT,
CAR.CONS, and CDR.CONS, and expanding the definition of
APPEND, to:

(TRUE).

Case 12.(IMPLIES

(AND

(NOT

(LESSP (SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*))
(SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYLEVEL) NIL)
(STATE*)))))

(NOT

(SUBSET (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYCATS) NIL)
(STATE*))
(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE SECURITYCATS) NIL)
(STATE*))
(STATE*)))))

(LESSP (SELECT (SELECT (LTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*))
(SELECT (SELECT (HTII*)
(CONS (QUOTE ND) NIL)
(STATE*))
(CONS (QUOTE INTEGRITYLEVEL) NIL)
(STATE*)))))

(NOT

(LESSP

(SELECT (HTII*)
(CONS (QUOTE ND)
(CONS (QUOTE SECURITYLEVEL) NIL))
(STATE*))
(SELECT (LTII*)
(CONS (QUOTE ND)
(CONS (QUOTE SECURITYLEVEL) NIL))
(STATE*)))))

(NOT

(LESSP

(SELECT
(LTII*)
(CONS (QUOTE ND)
(CONS (QUOTE INTEGRITYLEVEL) NIL))
(STATE*))

```

      (SELECT
        (HTII*)
        (CONS (QUOTE *)
          (CONS (QUOTE INTEGRITYLEVEL) NIL))
        (STATE*)))
    (SUBSET
      (SELECT (LTII*)
        (CONS (QUOTE NO)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (SELECT (HTII*)
        (CONS (QUOTE NO)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (STATE*)))
  (NOT
    (SUBSET
      (SELECT (LTII*)
        (CONS (QUOTE NO)
          (CONS (QUOTE SECURITYCATS) NIL))
        (STATE*))
      (SELECT (HTII*)
        (CONS (QUOTE NO)
          (CONS (QUOTE SECURITYCATS) NIL))
        (STATE*))
      (STATE*))))),
  (STATE*))))),

```

which we again simplify, rewriting with
 ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
 unfolding the function APPEND, to:

(TRUE).

Case 13.(IMPLIES

(AND

(NOT

(LESSP (SELECT (SELECT (HTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE SECURITYLEVEL) NIL)

(STATE*))

(SELECT (SELECT (LTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE SECURITYLEVEL) NIL)

(STATE*))))

(NOT

(SUBSET (SELECT (SELECT (LTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE SECURITYCATS) NIL)

(STATE*))

(SELECT (SELECT (HTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE SECURITYCATS) NIL)

(STATE*))

(STATE*))))

(NOT

(LESSP (SELECT (SELECT (LTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYLEVEL) NIL)

(STATE*))

(SELECT (SELECT (HTII*)

(CONS (QUOTE ND) NIL)

(STATE*))

(CONS (QUOTE INTEGRITYLEVEL) NIL)

(STATE*))))

(NOT

(LESSP

(SELECT (HTII*)

(CONS (QUOTE ND)

(CONS (QUOTE SECURITYLEVEL) NIL))

(STATE*))

(SELECT (LTII*)

(CONS (QUOTE ND)

(CONS (QUOTE SECURITYLEVEL) NIL))

(STATE*))))

(NOT

(LESSP

```

      (SELECT
        (LTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYLEVEL) NIL))
        (STATE*))
      (SELECT
        (HTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYLEVEL) NIL))
        (STATE*)))
    (SUBSET
      (SELECT (LTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (SELECT (HTII*)
        (CONS (QUOTE ND)
          (CONS (QUOTE INTEGRITYCATS) NIL))
        (STATE*))
      (STATE*)))
    (NOT
      (SUBSET
        (SELECT (LTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE SECURITYCATS) NIL))
          (STATE*))
        (SELECT (HTII*)
          (CONS (QUOTE ND)
            (CONS (QUOTE SECURITYCATS) NIL))
          (STATE*))
        (STATE*))))).

```

But this simplifies again, applying
 ASSOCIATIVITY.OF.SELECT, CAR.CONS, and CDR.CONS, and
 opening up the definition of APPEND, to:

(TRUE).

Q.E.D.

Load average during proof: 2.050366
 Elapsed time: 110.677 seconds
 CPU time (devoted to theorem proving): 43.741 seconds
 GC time: 9.887 seconds
 IO time: 7.267 seconds
 CUNSES consumed: 99301

PROVED

```

_UNDO.BACK.THROUGH(INPUT.TO.OUTPUT)
((PROVE.LEMMA CONCLUSION NIL (EQUAL (AND (AND (AND (NOT (
GREATERP (SELECT (SELECT (LTII*) (QUOTE (ND)) (STATE*)) (
QUOTE (SECURITYLEVEL)) (STATE*)) (SELECT (SELECT (HTII*) (
QUOTE (ND)) (STATE*)) (QUOTE (SECURITYLEVEL)) (STATE*))) (
SUBSET (SELECT (SELECT (LTII*) (QUOTE (ND)) (STATE*)) (QUOTE
(SECURITYCATS)) (STATE*)) (SELECT (SELECT (HTII*) (QUOTE (
ND)) (STATE*)) (QUOTE (SECURITYCATS)) (STATE*)) (STATE*)) (
NOT (LESSP (SELECT (SELECT (LTII*) (QUOTE (ND)) (STATE*)) (
QUOTE (INTEGRITYLEVEL)) (STATE*)) (SELECT (SELECT (HTII*) (
QUOTE (ND)) (STATE*)) (QUOTE (INTEGRITYLEVEL)) (STATE*))) (
SUBSET (SELECT (SELECT (LTII*) (QUOTE (ND)) (STATE*)) (
QUOTE (INTEGRITYCATS)) (STATE*)) (SELECT (SELECT (HTII*) (
QUOTE (ND)) (STATE*)) (QUOTE (INTEGRITYCATS)) (STATE*)) (
STATE*)) (AND (NOT (GREATERP (SELECT (LTII*) (QUOTE (ND
SECURITYLEVEL)) (STATE*)) (SELECT (HTII*) (QUOTE (ND
SECURITYLEVEL)) (STATE*))) (SUBSET (SELECT (LTII*) (QUOTE (
ND SECURITYCATS)) (STATE*)) (SELECT (HTII*) (QUOTE (ND
SECURITYCATS)) (STATE*)) (STATE*)) (NOT (LESSP (SELECT (
LTII*) (QUOTE (ND INTEGRITYLEVEL)) (STATE*)) (SELECT (HTII*)
(QUOTE (ND INTEGRITYLEVEL)) (STATE*))) (SUBSET (SELECT (
LTII*) (QUOTE (ND INTEGRITYCATS)) (STATE*)) (SELECT (HTII*)
(QUOTE (ND INTEGRITYCATS)) (STATE*)) (STATE*)))))) (COMMENT
INPUT.TO.OUTPUT))

```

```

_COMMENT(STATE.EQUIVALENCE)
STATE.EQUIVALENCE

```

```

_ADD.AXION(HYPOTHESIS
  (REWRITE)
  (AND (EQUAL (SUBSET X Y (NEWSTATE))
    (SUBSET X Y (STATE)))
    (EQUAL (SELECT STRUCTURE FIELD (NEWSTATE))
    (SELECT STRUCTURE FIELD (STATE)))))
HYPOTHESIS

```

```

_PROVE.LEMMA(CONCLUSION NIL
  (AND (EQUAL (SUBSET V1 V2 (NEWSTATE))
    (SUBSET V1 V2 (STATE)))
    (EQUAL (SELECT V1 V2 (NEWSTATE))
    (SELECT V1 V2 (STATE)))))

```

This conjecture can be propositionally simplified to two new formulas:

```

Case 1. (EQUAL (SUBSET V1 V2 (NEWSTATE))
  (SUBSET V1 V2 (STATE))).

```

This simplifies, applying the lemma HYPOTHESIS, to:

```

(TRUE).

```

Case 2. (EQUAL (SELECT V1 V2 (NEWSTATE))
(SELECT V1 V2 (STATE))),

which we simplify, rewriting with HYPOTHESIS, to:

(TRUE).

Q.E.D.

Load average during proof: 2.053366
Elapsed time: .989 seconds
CPU time (devoted to theorem proving): .342 seconds
GC time: 0.0 seconds
IO time: .303 seconds
CUNSES consumed: 261

PROVED

_UNDO.BACK.THROUGH(STATE.EQUIVALENCE)
((PROVE.LEMMA CONCLUSION NIL (AND (EQUAL (SUBSET V1 V2 (NEWSTATE)) (SUBSET V1 V2 (STATE))) (EQUAL (SELECT V1 V2 (NEWSTATE)) (SELECT V1 V2 (STATE)))) (ADD.AXIOM HYPOTHESIS (REWRITE) (AND (EQUAL (SUBSET X Y (NEWSTATE)) (SUBSET X Y (STATE))) (EQUAL (SELECT STRUCTURE FIELD (NEWSTATE)) (SELECT STRUCTURE FIELD (STATE)))) (COMMENT STATE.EQUIVALENCE))

_UNDO.BACK.THROUGH(CORRECTNESS.OF.SMXCOMPARE)
((DCL LTII*) (DCL HTII*) (COMMENT CORRECTNESS.OF.SMXCOMPARE))

_UNDO.BACK.THROUGH(
CORRECTNESS.OF.THE.IMPLEMENTATION.OF.SMXCOMPAREMODULE.ON.PRIMITIVEMODULE
)
((ADD.AXIOM ASSOCIATIVITY.OF.SELECT (REWRITE) (EQUAL (SELECT (SELECT S P1 STATE) P2 STATE) (SELECT S (APPEND P1 P2) STATE))) (ADD.AXIOM IDENTITY.OF.SELECT (REWRITE) (EQUAL (SELECT S NIL STATE) S)) (DCL SMXCOMPARE (LTII HTII STATE)) (DCL SELECT.RECORD (STRUCTURE FIELD STATE)) (DCL SUBSETOP (X Y STATE)) (DCL QUOTOP (X STATE)) (DCL ANDOP (X Y STATE)) (DCL OROP (X Y STATE)) (DCL LESSOREQUALOP (X Y STATE)) (DCL GREATEROREQUALOP (X Y STATE)) (DCL NUMBERPOP (X STATE)) (DCL DIFFERENCEOP (X Y STATE)) (DCL PLUSOP (X Y STATE)) (DCL ADDOP (X STATE)) (DCL LESSPOP (X Y STATE)) (DCL GREATERPOP (X Y STATE)) (DCL ZEROPOP (X STATE)) (DCL NEQUALOP (X Y STATE)) (DCL EQUALOP (X Y STATE)) (DCL UNDEFOP (STATE)) (DCL FALSEOP (STATE)) (DCL TRUEOP (STATE)) (DCL SELECT (STRUCTURE FIELD STATE)) (DCL SUBSET (X Y STATE)) (DCL NEXT (STATE)) (DCL NEWSTATE) (DCL BEGIN) (DCL STATE*) (DCL STATE) (COMMENT
CORRECTNESS.OF.THE.IMPLEMENTATION.OF.SMXCOMPAREMODULE.ON.PRIMITIVEMODULE
))

**DA
FILM**