Dave

CRMS PROCESS SYNCHRONIZATION MECHANISM
11 NOVEMBER 72
DAVE REDELL
PAUL MCJONES

I. MOTIVATION

THE FOLLOWING IS A DETAILED PROPOSAL FOR A FIRMWARE PROCESS SYNCHRONIZATION MECHANISM (PSM) FOR THE NEW SYSTEM. IT PROVIDES A REASONABLY UNIFORM PROCESS STRUCTURE FOR THE TWO PROCESSORS, WITH POWERFUL ENOUGH SIGNALING ACTIONS TO ALLOW A TRUSTED, COMPLETELY PRIVILEGED SOFTWARE "KERNEL" PROCESS TO BE CONSTRUCTED, WHICH CAN THEN EXTEND THE NOTIONS OF PROCESS AND SEGMENT, AND CAN ADD NOTIONS OF NEW OBJECTS (IN PARTICULAR, THE MESSAGE CHANNELS OF A FULL-FLEDGED IPC). TOGETHER, THE FIRMWARE AND THE KERNEL COMPOSE A SINGLE PROTECTED LAYER (PRESUMABLY LABELED "LAYER 1" IF THE HARDWARE IS "LAYER 0").

THREE ADVANTAGES ARE CLAIMED FOR THIS SCHEME AS COMPARED WITH MORE COMPLETE (AND, HENCE, MORE COMPLICATED) FIRMWARE MECHANISMS:

(1) CONCEPTUAL SIMPLICITY -- VERY LITTLE IS ASSUMED ABOUT THE DETAILS OF THE MORE COMPLICATED ASPECTS OF INTERPROCESS COMMUNICATION. THESE SUBTLE DETAILS CAN BE MORE EASILY IMPLEMENTED AND MODIFIED IF DONE IN SOFTWARE.

(2) SMALL SIZE -- THE MICROPROGRAMS TO IMPLEMENT THE PSM WILL UNDOUBTEDLY FIT IN THE REMAINING ROM, WITH SOME ROOM LEFT OVER FOR IPU PATCHING AND OTHER IMPONDERABLES. ANY SIGNIFICANTLY MORE COMPLICATED SCHEME IS LIKELY TO REDUCE THE UNUSED ROM SPACE TO AN UNACCEPTIBLE LEVEL (IF NOT ELIMINATE IT COMPLETELY).

(3) APPROPRIATE DIVISION OF TASKS -- IT IS TEMPTING TO REGARD THE FIRMWARE-SOFTWARE INTERFACE AS A LOGICAL LAYER BOUNDARY, BUT IT MUST BE REMEMBERED THAT IT IS, FIRST AND FOREMOST, A TECHNOLOGICAL BOUNDARY. THUS, ASSIGNMENT OF A TASK TO FIRMWARE MUST BE MADE PRIMARILY ON THE BASIS OF FREQUENCY OF USE. ON THE OTHER HAND, IF THE FIRMWARE WERE TO FORM A PROTECTED LAYER WHOSE INTEGRITY WERE TO BE GUARANTEED AGAINST ALL SOFTWARE ACTIONS, THEN LOGICAL COMPLETENESS WOULD REQUIRE FIRMWARE IMPLEMENTATION OF ANY ACTION WHICH MANIPULATED A DATA BASE TRUSTED BY THE FIRMWARE LAYER, NO MATTER HOW INFREQUENTLY THAT ACTION MIGHT BE INVOKED (FOR EXAMPLE, "CREATE CAPABILITY SEGMENT").

FOR THESE REASONS, THE PSM AS DESCRIBED BELOW IS PROPOSED. WHILE LACKING SOPHISTICATION AND, PERHAPS IN SOME SITUATIONS, A CERTAIN AMOUNT OF EFFICIENCY, IT IS SIMPLE AND RATHER CLEAN.

II. PROCESSES

PROCESSES ARE THE ONLY ACTIVE ENTITIES IN THE SYSTEM; EACH PROCESSOR EXECUTES INSTRUCTIONS ONLY ON BEHALF OF ITS CURRENT PROCESS. AT ANY GIVEN TIME, A PROCESS IS SAID TO BE READY IF IT IS "LOGICALLY RUNNING" (I.E. WOULD BE RUNNING IF A PHYSICAL PROCESSOR WERE AVAILABLE). OTHERWISE, IT IS SAID TO BE NOT-READY.

THE ACTUAL STATES OF A PROCESS ARE A LITTLE MORE COMPLICATED, CONSISTING OF TWO COMPONENTS. ONE COMPONENT IS ORIENTED TOWARD EXTERNAL CONTROL OF ONE PROCESS BY ANOTHER, AND CONSIDERS EACH PROCESS TO BE ON OR OFF. THE OTHER COMPONENT IS ORIENTED TOWARD VOLUNTARY COOPERATION BETWEEN PROCESSES, AND CONSIDERS EACH PROCESS TO BE BLOCKED, AWAKE, OR AWAKE-WITH-WAKEUP-WAITING. THE STATE SET OF A PROCESS IS THUS THE CROSS-PRODUCT OF THESE TWO COMPONENT STATE-SETS (SEE FIGURE 1).

THE RELATION BETWEEN THE STATE-SET AND THE NOTIONS OF READY AND NOT-READY IS NOW QUITE STRAIGHTFORWARD: A PROCESS WHICH IS ON AND AWAKE (WITH OR WITHOUT WAKEUP-WAITING) IS READY. A PROCESS WHICH IS OFF OR BLOCKED IS NOT-READY (AGAIN, SEE FIGURE 1).

A PROCESS MAY CHANGE STATE AS A RESULT OF ITS OWN ACTIONS, OR THOSE OF ANOTHER PROCESS. THE RELEVANT PROCESSOR (FIRMWARE) OPERATIONS ARE:

(1) ON(P)
 TURNS PROCESS P ON.
 (ERROR IF P IS ALREADY ON.)
(2) OFF(P)
 TURNS PROCESS P OFF.
 (ERROR IF P IS ALREADY OFF.)
(3) BLOCK(P)
 MAKES PROCESS P BLOCKED.
 (UNLESS WAKEUP WAITING -- SEE FIGURE 1.)
(4) WAKEUP(P)
 MAKES PROCESS P AWAKE.
 (UNLESS ALREADY AWAKE -- SEE FIGURE 1.)
(5) ATTENTION TRAP(N)
 CAUSES A TRAP OF CLASS "ATTENTION", NUMBER N.
 (SEE BELOW FOR DISCUSSION OF TRAPS.)

NOTE A) IN CASES 1 THROUGH 4, P IS A PROCESS CAPABILITY POSSESSING SUITABLE PRIVILEGES.
 B) THUS ONLY ACTION 5 IS AVAILABLE TO APL PROCESSES.

III. TRAPS

A TRAP IS AN AUTOMATIC WAKEUP WHICH IS DIRECTED TO A SPECIAL PROCESS KNOWN TO THE PSM FIRMWARE (I.E. THE KERNEL). WHEN CERTAIN CONDITIONS ARISE DURING THE COURSE OF EXECUTION OF A NORMAL PROCESS, SUCH A WAKEUP IS GENERATED, AND A TRAP DATUM IS STORED IN A FIXED MEMORY LOCATION ASSOCIATED WITH THE PROCESSOR. THE TRAP DATUM IDENTIFIES:

(1) THE TRAPPED PROCESS (I.E. UNIQUE NAME);
(2) THE CONDITION WHICH CAUSED THE TRAP (I.E. CLASS AND NUMBER -- SEE BELOW).

THE KERNEL (BEING THE HIGHEST PRIORITY PROCESS IN THE SYSTEM) IMMEDIATELY BEGINS EXECUTION TO PROCESS THE TRAP. BY READING THE TRAP DATUM (AND POSSIBLY ADDITIONAL DATA FROM THE MEMORY OF THE TRAPPED PROCESS), THE KERNEL CAN DECIDE ON THE APPROPRIATE ACTION.

GENERALLY, A TRAP IS GENERATED BY SOME APPARENT ERROR DETECTED BY THE PROCESSOR (E.G. OVERFLOW, MISSING SEGMENT, ETC.). IN ADDITION, A TRAP CAN BE GENERATED BY THE ATTENTION TRAP OPERATION. FOR THE CONVENIENCE OF THE KERNEL, EACH TYPE OF TRAP IS GIVEN A TWO-PART NAME: <CLASS, NUMBER>. THE CLASS IDENTIFIES THE BASIC NATURE OF THE TRAP (E.G. ATTENTION, ARITHMETIC, MEMORY, INSTRUCTION, ETC.), WHILE THE NUMBER SPECIFIES EXACTLY WHAT CAUSED THE TRAP (DIVISION BY ZERO, APL DOMAIN ERROR, REFERENCE OUTSIDE SEGMENT, ETC.).

EACH PROCESSOR HAS A SINGLE TRAP DATUM LOCATION. TO AVOID THE DESTRUCTION OF A TRAP DATUM BY A SECOND IMMEDIATELY FOLLOWING TRAP ON THE SAME PROCESSOR, A FEW RESTRICTIONS ARE NEEDED:

(1) THE KERNEL MUST BE THE HIGHEST PRIORITY SIMPLE PROCESS;
(2) THE APL PROCESSOR, UPON GENERATING A TRAP, MUST ENTER AN "IDLE" STATE UNTIL A "PROCEED" SIGNAL IS RECEIVED FROM THE SIMPLE PROCESSOR (ACKNOWLEDGING RECEIPT BY THE KERNEL OF THE TRAP DATUM). FORTUNATELY, THIS EXACT MECHANISM IS ALREADY PRESENT AS A LOW LEVEL ROUTINE IN THE READYLIST FIRMWARE (SEE BELOW), SO NO EXTRA COMPLICATION IS BEING INTRODUCED.

WHEN A TRAP IS GENERATED, THE TRAPPED PROCESS IS LEFT READY. IF THE KERNEL CAN HANDLE THE TRAP IN ONE CONTIGUOUS COMPUTATION AND THEN BLOCK, THE TRAPPED PROCESS CAN CONTINUE WITHOUT EVER HAVING LEFT THE READYLIST. IF, ON THE OTHER HAND, THE KERNEL MUST BLOCK DURING THE COURSE OF HANDLING THE TRAP (E.G. I/O FOR MISSING SEGMENT TRAP), IT CAN FIRST TURN OFF THE TRAPPED PROCESS BEFORE TRAP HANDLING IS COMPLETED. NOTE THAT "SYSTEM CALLS", BEING ATTENTION TRAPS, ALSO FIT INTO THIS SCHEME.

THUS, TRAPS AND SYSTEM CALLS ARE DIVIDED INTO "LIGHTWEIGHT" AND "HEAVYWEIGHT". LOGICALLY, NO DIFFERENCE IS APPARENT TO THE CALLER, BUT IN TERMS OF SCHEDULING, CERTAIN ABERRATIONS ARE AVOIDED (E.G. TWO PROCESSES OF THE SAME PRIORITY GETTING DIFFERENT FRACTIONS OF THE PROCESSOR BECAUSE ONE DOES MANY MORE LIGHTWEIGHT SYSTEM CALLS).

TRAPS ALWAYS LEAVE THE PROGRAM-COUNTER OF THE TRAPPED PROCESS POINTING BEFORE THE INSTRUCTION WHICH CAUSED THE TRAP. IF THIS INSTRUCTION IS NOT TO BE RE-TRIED, THE TRAP HANDLING SOFTWARE MUST INCREMENT THE PROGRAM-COUNTER BEFORE ALLOWING THE TRAPPED PROCESS TO PROCEED. THIS CONVENTION IS FAIRLY IMPORTANT, SINCE, IN GENERAL, DECREMENTING THE PROGRAM-COUNTER WOULD BE DIFFICULT, IF NOT IMPOSSIBLE, DUE TO VARIABLE LENGTH INSTRUCTIONS.

ONE MAY ASK, WHY DO WE NEED BOTH FIRMWARE OFF-ON AND BLOCK-WAKEUP MECHANISMS? THE ANSWER IS THAT WAKEUPS WILL BE USED FOR I/O INTERACTIONS AND TRAP HANDLING. IN EACH CASE, THE WAKEUP-WAITING LOGIC IS NEEDED. FURTHERMORE, WE WANT TO MAKE WAKEUPS AVAILABLE AS PROCESSOR INSTRUCTIONS FOR TWO REASONS:

(1) TO ALLOW TESTING OF I/O AND TRAP PROGRAMS BEFORE PUTTING THEM INTO THE SYSTEM;
(2) TO ALLOW VERY HIGH SPEED INTERACTION OF CLOSELY COUPLED SYSTEM PROCESSES.

NOW, WE NEED THE ON-OFF STUFF TOO, FOR THE MESSAGE HANDLER. IF THE WAKEUP MECHANISM WERE USED HERE, EACH PROCESS WOULD HAVE TO BE IN ONE OF TWO CLASSES (TO AVOID CONFLICTING USE OF THE WAKEUP MECHANISM):

A) PROCESSES WHICH USED THE WAKEUP MECHANSIM DIRECTLY, AND COULD NOT USE THE MESSAGE CHANNELS AT ALL;
B) PROCESSES WHICH USED MESSAGE CHANNELS, AND HENCE COULD NOT DIRECTLY USE THE FIRMWARE WAKEUP MECHANISM.

THERE ARE TWO PROBLEMS WITH THIS SCHEME:

(1) IT WOULD BE NICE TO ALLOW A GIVEN PROCESS TO USE BOTH MECHANISMS, RATHER THAN HAVING TO CHOOSE ON OR THE OTHER;
(2) IT WOULD BE DIFFICULT (ALTHOUGH PROBABLY POSSIBLE) FOR PROCESSES IN THESE TWO CLASSES TO COMMUNICATE WITH EACH OTHER. (SOMEHOW HAVE MESSAGE-HANDLER PROVIDE "APPROVED" WAKEUP AND BLOCK OPERATIONS. VERY COMPLICATED (E.G. TYPE A PROCESS WAKES UP A TYPE B PROCESS USING CPU OPERATION). ALSO, TRAPS WOULD BE A MESS.)

IV. IMPLEMENTATION DETAILS

THE FIRMWARE FOR IMPLEMENTING THE PROCESS SYNCHRONIZATION MECHANISM BREAKS DOWN RATHER CLEANLY INTO TWO LEVELS:

A) A HIGH LEVEL, IMPLEMENTING THE PROCESSOR INSTRUCTIONS AND TRAPS AS DESCRIBED ABOVE;
B) A LOW LEVEL, WHICH MAINTAINS A READYLIST FOR EACH PROCESSOR AND ASSURES THAT EACH PROCESSOR IS ALWAYS RUNNING THE HIGHEST PRIORITY PROCESS ON ITS READYLIST.

OTHER MICROPROGRAMS OUTSIDE THE PSM USE THE HIGH LEVEL WAKEUP FIRMWARE (E.G. I/O CONTROLLERS, PROCESSOR TRAP GENERATION ROUTINES, QUANTUM-OVERFLOW TRAP MACHINERY, ETC.). THE LOW LEVEL ROUTINES ARE USED ONLY BY THE HIGH LEVEL ROUTINES.

THE VARIOUS PROCESSOR INSTRUCTIONS CAUSE STATE TRANSITIONS OF THE CURRENT PROCESS, THE PROCESS SPECIFIED AS ARGUMENT TO THE OPERATION, OR THE SPECIALLY KNOWN (KERNEL) PROCESS. IN EACH CASE, THE TRANSITION MAY OR MAY NOT IMPLY A CALL ON THE LOW LEVEL ROUTINES TO INSERT OR REMOVE THE AFFECTED PROCESS IN THE READYLIST. IT IS CLEAR FROM FIGURE 1 WHICH TRANSITIONS THESE ARE: NAMELY THOSE WHICH CROSS THE BOUNDARY BETWEEN READY STATES AND NOT-READY STATES.

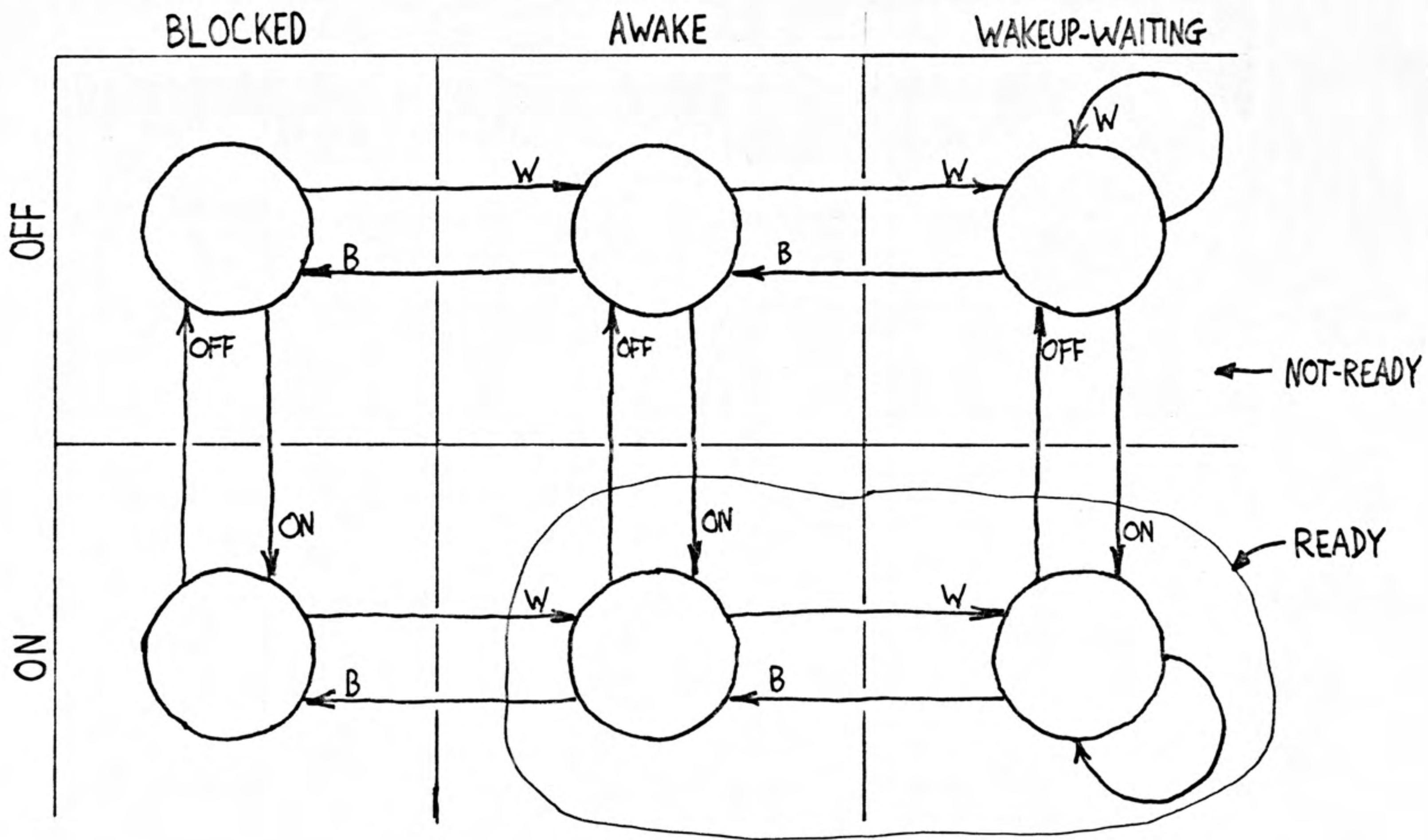
FOLLOWING ARE ALGORITHMS FOR THE INSTRUCTIONS OFF, ON, BLOCK, AND WAKEUP, WRITTEN IN PSEUDO-SIMPLE.

```
FUNCTION OFF( CAPAB P ):
*
* TURN OFF PROCESS P
*
      DECLARE REFERENCE PPTE;
      *
      IF P.TYPE=PROCESS THEN:
            IF HASPRIVILEGE( P, ONOFF ) THEN:
                  PTE ← @PROCTAB[ P.UNIQUENAME ];
                  IF PTE.CTRLSTATE=ON THEN;
                        PTE.CTRLSTATE ← OFF;
                        IF PTE.COOPSTATE#BLOCKED THEN;
                              UNREADY( PTE );
                        ENDIF;
                  ELSE:
                        TRAP( PROCESS, OFFOFF );
                  ENDIF;
            ELSE:
                  TRAP( CAPABILITY, NOPRIVILEGE );
            ENDIF;
      ELSE;
            TRAP( CAPABILITY, BADTYPE );
      ENDIF;
      RETURN;
ENDFUNCTION;
```

```
FUNCTION ON( CAPAB P );
*
* TURN ON PROCESS P
*
    DECLARE REFERENCE PTE;
    *
    IF P.TYPE=PROCESS THEN;
        IF HASPRIVILEGE( P, ONOFF ) THEN;
            PTE ← @PROCTAB[ P.UNIQUENAME ];
            IF PTE.CTRLSTATE=OFF THEN;
                PTE.CTRLSTATE ← ON;
                IF PTE.COOPSTATE#BLOCKED THEN;
                    READY( PTE );
                ENDIF;
            ELSE;
                TRAP( PROCESS, ONON );
            ENDIF;
        ELSE;
            TRAP( CAPABILITY, NOPRIVILEGE );
        ENDIF;
    ELSE;
        TRAP( CAPABILITY, BADTYPE );
    ENDIF;
    RETURN;
ENDFUNCTION;
```

```
FUNCTION BLOCK(P):
*
* MAKE PROCESS P BLOCKED
*
     DECLARE REFERENCE PTE:
     *
     IF P.TYPE=PROCESS THEN;
          IF HASPRIVILEGE( P, BLOCKWAKEUP ) THEN;
               PTE ← @PROCTAB[ P.UNIQUENAME ]:
               IF PTE.COOPSTATE=WAKEUPWAITING THEN;
                    PTE.COOPSTATE ← AWAKE:
               ELSEIF PTE.COOPSTATE=AWAKE THEN;
                    PTE.COOPSTATE←BLOCKED:
                    IF PTE.CTRLSTATE=ON THEN;
                         UNREADY( PTE );
                    ENDIF:
               ELSE;
                    TRAP( PROCESS, BLOCKBLOCKED );
               ENDIF;
          ELSE:
               TRAP( CAPABILITY, NOPRIVILEGE );
          ENDIF:
     ELSE:
          TRAP( CAPABILITY, BADTYPE ):
     ENDIF:
     RETURN:
ENDFUNCTION;
```

```
FUNCTION WAKEUP( CAPAB P );
*
* WAKE UP PROCESS P
*
      DECLARE REFERENCE PTE;
      *
      IF P.TYPE=PROCESS THEN;
            IF HASPRIVILEGE( P, BLOCKWAKEUP ) THEN;
                  PTE ← @PROCTAB[ P.UNIQUENAME ];
                  IF PTE.COOPSTATE=AWAKE THEN;
                        PTE.COOPSTATE ← WAKEUPWAITING;
                  ELSEIF PTE.COOPSTATE=BLOCKED THEN;
                        PTE.COOPSTATE ← AWAKE;
                        IF PTE.CTRLSTATE=ON THEN;
                              READY( PTE );
                        ENDIF;
                  ENDIF;
            ELSE;
                  TRAP( CAPABILITY, NOPRIVILEGE );
            ENDIF;
      ELSE;
            TRAP( CAPABILITY, BADTYPE );
      ENDIF;
      RETURN;
ENDFUNCTION;
```



B = BLOCK
W = WAKEUP

OFF = TURN OFF
ON = TURN ON

Figure 1