A Teeny-tiny Operating System

Paul R. Potts

1987

In my sophomore year of college I took two interesting classes that helped "make my bones" as a programmer. One was called Assembly Language, and one was called Operating Systems. In the assembly language class we wrote a series of assignments that culminated in writing a (simplified) assembler for the VAX 11/750, in VAX 11/750 assembly language. In the operating systems class, we read *Operating System Concepts, Second Edition* by Peterson and Silberschatz and studied the theory behind operating systems. But we also had a "practicum" — we implemented an operating system.

Well, sort of. We weren't Linus Torvalds writing Linux back then. We had a small lab set up with Motorola 68000 "trainer" development boards, each of which contained the same microprocessor that was in the original Apple Macintosh. We used a cross-assembler that ran on the VAX 11/750 to build 68000 assembly-language code for this board, download it, and run it. There was a little low-level "monitor" program in ROM on this board that made this possible.

Why the 68000? We and our instructor were quite interested in the 68000 at the time, as it was a very capable microprocessor, with a larger memory space and wider data bus than the earlier 6502 used in the Apple II or 8088 used in the original IBM PC. Our instructor Simon Tung was especially interested in the illegal opcode exception-handling, as this was used by Apple to implement the early Macintosh Toolbox. I vividly recall him drawing a diagram of the exception-handling logic at the gate level.

The exception-handling logic provided a fast and efficient way to trigger an exit from user code to privileged system code in ROM. This saved a lot of RAM by keeping most of the toolbox code in ROM — remember, RAM was scarce and expensive back then compared to ROM. But this mechanism worked by using a jump table stored in RAM, which allowed code on disc to patch operating system calls, both to fix bugs and to do other neat tricks. I would eventually teach how to write a variety of programs for old MacOS in C and Pascal, and that became a big part of my early career, but that's another story.

We began with some very simple low-level functions, but the final assignment was to put them together into a "teeny-tiny operating system." It was a program

that ran a timer and read keyboard input using interrupts, launched a number of tasks, and switched periodically between tasks. So we'd press a key, and the program would start spitting out that character to the terminal at a specific rate. We'd press additional keys, and it would start more tasks to do the same thing with different characters at different rates. We could kill processes. This functionality is what an operating system does. Of course, it's not *all* a modern operating system does.

I don't recall all the details (it's been almost 40 years, after all), but I recall that I had my program working perfectly, with beautiful timing, but I screwed something up, and I was never able to get the timing to work quite that well again. Oh well. If you spot the bug, let me know!

Note that the web version of the code is a bit ugly, because my web page template imposes a strict width restriction on preformatted text, which results in the ends of some comments getting cut off (although you can scroll right to read them). Until I can fix that, I recommend reading the PDF version of this file.

```
* Original filename: as3.3.src;
               LLEN            120             ;ALIGN NICELY

*****************************************************************
***********    ASSIGNMENT 3 PART 3 FOR CS 256 BY PAUL POTTS
***********    THIS PROGRAM IMPLEMENTS UP TO 16 PROCESSES CALLED
***********    'USER' WHICH SIMPLY PRINT OUT A SPECIFIC CHARACTER
***********    STORED IN THE PCB.  THE DELAYS FOR NEW PROCESSES
***********    START OUT BEING 1000 UNITDELAYS OF .001 SECONDS,
***********    OR 1 SECOND. IN ORDER TO KEEP THE PROCESSES
***********    FROM OVERFLOWING EVEN A LARGE QUEUE THE SMALLEST DELAY
***********    ALLOWED MUST BE 5 UNITDELAYS.  THE USER CAN CHANGE THE DELAY
***********    BY STRIKING '+' TO DOUBLE IT OR '-' TO CUT IT
***********    IN HALF WHILE THE PROCESS IS RUNNING.  THE KEY
***********    '#' WILL TERMINATE EXECUTION OF THE PROCESS.
*********** THE SYSTEM STARTS OUT RUNNING NO PROCESSES AND GENERATING NO
*********** TIMER INTERRUPTS TO SWAP PROCESSES.  PROCESSES ARE ADDED TO THE
*********** SYSTEM BY STRIKING KEYS OTHER THAN THE ONES MENTIONED ABOVE.
*********** THESE PROCESSES WILL CONTINUE TO CYCLE UNTIL THEY ARE KILLED.
*********** NOTE THAT IF THE TIME DELAY OF A PROCESS IS MADE SMALL ENOUGH
*********** IT WILL PRINT SEVERAL REPETITIONS OF ITS CHARACTER IN ITS
*********** TIME SLICE, SINCE IT WILL CALL "DISPLAY" SEVERAL TIMES
*********** BEFORE IT IS SWAPPED OUT.
*****************************************************************
* THE FOLLOWING ARE SYSTEM-DEPENDENT LABELS GIVEN TO THE SPECIAL
* REGISTERS OF THE 68000 BOARD
*****************************************************************

ACIAVECTOR      EQU             4*29    ;VECTOR FOR ACIA INTERRUPT
```

```
TIMERVECTOR     EQU             4*26    ;VECTOR FOR TIMER INTERRUPT
ALINEVECTOR     EQU             4*10    ;VECTOR FOR A-LINE HANDLER
TCR             EQU             $10021          ;TIMER CONTROL REGISTER
TIV             EQU             $10023          ;TIMER INTERRUPT VECTOR
CPR             EQU             $10025          ;COUNTER PRELOAD AREA
ACIACS          EQU             $10040          ;CONTROL/STATUS REGISTER
ACIAD           EQU             $10042          ;DATA REGISTER
ACIAMODE        EQU             15              ;THE MODE WANTED
QSIZE           EQU             40              ;40 SPACES IN THE QUEUE
SRSTART         EQU             $2000           ;INITIAL SR OF EACH PROCESS
DELAY1          EQU             100             ;DELAY OF PROCESS=.1 SEC.
NEWSLICE        EQU             125000          ;STARTING SLICE=1 SEC.

*****************************************************************

TOP             ORG             $1000           ;START HERE
                JMP             START           ;BRANCH DOWN
TIMESLICE       DS.L            1               ;START TIMESLICE AT 10 SEC.

*************************************************************************
**** THIS DEFINITION OF THE QUEUE DOES NOT USE INTERNAL LABELS BUT JUST
**** A POINTER TO THE WHOLE QUEUE.  THUS, QUEUE POINTS TO THE HEAD OFFSET,
**** QUEUE+2 POINTS TO THE TAIL OFFSET, QUEUE+4 POINTS TO THE LENGTH, AND
**** QUEUE +6 IS THE FIRST BYTE OF THE QUEUE.  THE QUEUE STORAGE TAKES
**** LENGTH OF QUEUE PLUS ONE BYTES, ALL BUT ONE OF WHICH CAN HOLD ACTUAL
**** DATA.  THE EXTRA IS USED TO FIGURE OUT WHEN THE QUEUE IS FULL.
*************************************************************************

QUEUE           DS.W            1               ;THE HEAD POINTER
                DS.W            1               ;THE TAIL POINTER
                DC.W            QSIZE           ;THE QUEUE LENGTH
                DS.B            QSIZE+1         ;THE LENGTH PLUS AN EXTRA

*****************************************************************
* PCOUNT HOLDS THE NUMBER OF ACTIVE PROCESSES, FROM ZERO TO 16
*****************************************************************

PCOUNT          DS.B            1               ;NUMBER OF PROCESSES

*****************************************************************
* THIS IS THE STRUCTURE OF A PCB IN MEMORY:
*
* storage   holds
* BYTE:     ASCII ID CHARACTER FOR THE PROCESS
* BYTE:     FOR FUTURE USE
* WORD:     FOR HOLDING THE STATUS REGISTER
```

```
* LONGWORD: PC
* LONGWORD: D0
* LONGWORD: D1
* LONGWORD: D2
*
* TOTAL PCB: EACH TAKES 20 BYTES
*
*********************************************************************

PCBTOP          DS.B    320     ;SPACE FOR 30 PCBS

PCBPOINTER      DS.L    1       ;POINTER TO THE CURRENT PCB

*****************************************************
* CODE TO ALLOW OR DISALLOW THE HANDLING OF INTERRUPTS
*****************************************************

UNBLOCKINT      MACRO
                ANDI    #$F8FF,SR       ;ENABLE INTERRUPTS
                ENDM

*****************************************************

BLOCKINT        MACRO
                ORI     #$0700,SR       ;DISABLE INTERRUPTS
                ENDM

*****************************************************
* DISPLAY FORCES AN A-LINE EXCEPTION TO OCCUR, AND THE
* SYSTEM JUMPS TO THE CODE BELOW, AHANDLER, WHEN IT OCCURS.
*****************************************************

DISPLAY         MACRO
                DC.W    $A000           ;A-LINE FOR DISPLAY
                ENDM

*************************************************************
* THE A-LINE HANDLER
* THIS ROUTINE SIMPLY COPIES THE CURRENT PROCESS ID CHARACTER
* INTO D7, THEN CALLS ENQ TO PUT IT IN THE QUEUE.
* THE REGISTERS A6, D0, D1, AND D2 ARE NOT DESTROYED, BUT
* OTHERS ARE USED IN THE QUEUE ROUTINE.
*************************************************************

AHANDLER        BLOCKINT
                MOVEA.L PCBPOINTER,A5           ;ADDRESS POINTED TO
```

```
                MOVE.B  (A5),D7                 ;PUT ID CHAR INTO D7
                JSR     ENQ                     ;ENQUEUE THE BYTE
                ADD.L   #2,$2(A7)               ;UPDATE OLD PC
                RTE

*************************************************************
* SUBROUTINE:  INITALINE
* THIS ROUTINE SIMPLY DOES INITIALIZATION OF THE A-LINE EMULATOR.
*************************************************************

INITALINE       MOVE.L  #AHANDLER,ALINEVECTOR           ;A-LINE VECTOR
                RTS

*************************************************************************
* SUBROUTINE:  INITPCBS
* THIS ROUTINE SETS UP THE INITIAL VALUES OF THE THREE PCBS IN MEMORY.
* THIS IS WRITTEN AS CODE INSTEAD OF SIMPLY FILLING MEMORY WITH
* DC.W OR DC.L SO THAT THE PROGRAM CAN BE RESTARTED WITHOUT LOSING
* THE INITIAL CONTENTS OF THE PCBS.
*
* REGISTERS USED: A0 AND D4 ONLY.  THE ORIGINAL VALUES ARE RESTORED
*************************************************************************

INITPCBS        MOVEM.L D4/A0,-(A7)             ;PUSH REGISTERS
                MOVEA.L #PCBTOP,A0              ;HOLD TEMPORARILY
                MOVE.L  A0,PCBPOINTER           ;SET UP INITIAL VALUE
                MOVE.B  #0,PCOUNT               ;NUMBER OF PROCESSES=0
                MOVE.L  #15,D4                  ;SET UP LOOP COUNTER

COUNTLOOP       MOVE.B  #0,(A0)                 ;SET ID/OUTPUT CHAR TO NULL
                ADD.L   #2,A0                   ;POINT TO SR
                MOVE.W  #SRSTART,(A0)           ;COPY INTO PCB
                ADD.L   #2,A0                   ;POINT TO FIRST PC
                MOVE.L  #WHILELOOP,(A0)         ;COPY INTO PCB
                ADD.L   #16,A0                  ;POINT TO NEXT PCB
                DBEQ    D4,COUNTLOOP            ;BUILD NEXT PCB
                MOVEM.L (A7)+,D4/A0             ;POP REGISTERS
                RTS

*************************************************************************
****** THIS ROUTINE IS MOD:  IT ACCEPTS THE OFFSET STORED IN A WORD
****** (OF 1..QSIZE + 1) AND RETURNS THE REAL MEMORY ADDRESS TO PUT
****** OR GET THE DATA BYTE.  IT SERVES TO WRAP AROUND THE QUEUE SO
****** THAT IT ISN'T POSSIBLE TO WALK OFF THE BOTTOM OF THE DATA
****** STRUCTURE BY INCREMENTING YOUR OFFSETS TOO MUCH.  IF THE POINTER
****** IS TOO LARGE (MEANING THAT IT CONTAINS QSIZE+1) THE MACRO
```

```
****** RESETS IT TO ZERO, OR THE BEGINNING OF THE QUEUE.
*************************************************************************
* The first parameter is the word offset
* The second parameter is the real address of where the byte is or should go

MOD             MACRO
                MOVEM.L D1-D2/A4,-(A7)  ;PUSH STUFF ON
                MOVEA.L A6,A4           ;START AT TOP OF QUEUE
                MOVE.W  \1,D1           ;HOLD THE VALUE OF THE OFFSET
                MOVE.W  #0,\1           ;SET THE OLD OFFSET FOR WRAPAROUND
                ADD.L   #6,A4           ;POINT TO QUEUE DATA
                MOVE.L  A4,\2           ;POINT TO THE START OF QUEUE DATA
                CMP.W   #QSIZE,D1       ;IS IT WITHIN 1..QSIZE?
                BGT     \@              ;SHOULD WE WRAP THE BYTE AROUND?
                MOVE.W  D1,\1           ;RESTORE THE VALUE OF THE POINTER
                ADD.W   \1,\2           ;OR INCLUDE OFFSET

* THIS CASE IS EXECUTED IF THE QUEUE HAS WRAPPED AROUND.  THE POINTER
* IN \1 IS THEN LEFT AT ZERO (AS IT WAS SET ABOVE) AND THE ADDRESS IS
* LEFT AT THE START OF THE QUEUE DATA (AS IT WAS SET IN A4.)

\@              MOVEM.L (A7)+,D1-D2/A4  ;POP STUFF OFF
                ENDM

*************************************************************
*
* THE PURPOSE OF THIS MACRO IS TO CONSUME .001 SECONDS BY
* LOOPING 280 TIMES.  THE PASCAL EQUIVALENT CODE IS:
*
* IT USES REGISTER D2 TO COUNT DOWN AND THUS DESTROYS ITS CONTENTS
*
* PROCEDURE UNITDELAY;
* VAR X: INTEGER;
* BEGIN
*       FOR X:=280 DOWNTO 1 DO {NOTHING};
* END;
*
*************************************************************

UNITDELAY       MACRO
                MOVE.L  #280,D2         ;COUNT DOWN FROM 280
\@              DBEQ    D2,\@           ;LOOP IF NOT DONE
                ENDM

*************************************************************
* THIS MACRO MULTIPLIES THE USER'S T BY TWO.  IT CAN AVOID THE
```

```
* LIMITS OF WORD MULTIPLICATION BY SHIFTING BY ONE BIT
* INSTEAD.  THE USER'S T IS PASSED IN D0.  THIS MACRO ALSO
* ALLOWS YOU TO INCREASE YOUR TIME DELAY EVEN IF DIVIDE_T
* HAS SET IT TO ZERO, SO THAT YOU CAN'T BECOME "STUCK."
* IT AFFECTS NO OTHER REGISTERS.
*****************************************************************

MULTIPLY_T      MACRO
                ASL.L   #1,D0                   ;MULTIPLY D0 BY 2
                ENDM

*****************************************************************
* THIS IS WRITTEN AS A MACRO TO AVOID PROBLEMS WITH THE REMAINDER
* IN THE UPPER WORD OF D0 AND TO SIMPLIFY THE MAIN PROCEDURE.
* THE MACRO SIMPLY DIVIDES THE USER'S DELAY BY TWO.
* IT USES NO OTHER REGISTERS THAN D0 (THE USERS'S DELAY)
*****************************************************************

DIVIDE_T        MACRO
                ASR.L   #1,D0           ;DIVIDE D0 BY 2
                CMP.L   #5,D0           ;IS DELAY TOO SMALL?
                BGT     \@              ;IF NOT, SKIP DOWN
                MOVE.L  #5,D0           ;SET IT TO FIVE.
\@              NOP                     ;FINISH MACRO
                ENDM

*****************************************************************************
****** THIS SUBROUTINE SETS UP THE TIMER TO BEGIN GENERATING
****** INTERRUPTS EVERY (TIMESLICE) CYCLES--WHEN AN INTERRUPT
****** OCCURS IT WILL TRAP TO THE CONTENTS OF VECTOR 15
****** TIMESLICE CAN BE ALTERED BY THE PERSON RUNNING THE PROGRAM.
* NOTE THAT TIMERINIT NO LONGER ENABLES THE TIMER: THIS IS DONE
* WHEN PROCESSES ARE ADDED SO THAT TIMER INTERRUPTS DO NOT OCCUR
* WHEN NO PROCESSES ARE RUNNING IN THE SYSTEM.
* IT CHANGES NO REGISTERS PERMANENTLY.
*****************************************************************************

TIMERINIT       MOVEM.L A0/D0,-(A7)             ;PUSH
                MOVE.L  #NEWSLICE,TIMESLICE     ;INITIAL TIMESLICE
                MOVE.L  TIMESLICE,D0            ;NUMBER OF CYCLES
                MOVEA.L #CPR,A0                 ;COUNTER PRELOAD REGISTERS
                MOVEP.L D0,0(A0)                ;HAVE TO USE THIS INSTRUCTION
                MOVE.B  #26,TIV                 ;AUTOVECTOR NUMBER
                MOVE.L  #MULTIPLEXER,TIMERVECTOR        ;VECTOR FOR TIMER
                MOVEM.L (A7)+,A0/D0             ;POP
                RTS                             ;RETURN
```

```
*************************************************************************
****** THIS SUBROUTINE INITIALIZES THE ACIA1 AND SETS UP THE VECTOR
****** SO THAT WHEN AN INTERRUPT OCCURS THE EXCEPTION HANDLER ACIAHANDLER
****** WILL BE CALLED TO DETERMINE WHETHER IT IS RECEIVE (FROM THE SCREEN),
****** TRANSMIT (FROM THE KEYBOARD) OR OVERRUN THAT IS CAUSING THE INTERRUPT.
****** WHEN AN INTERRUPT OCCURS FROM THE ACIA IT TRAPS THROUGH THE 29TH
****** VECTOR OF THE TABLE.  IT CHANGES NO REGISTERS PERMANENTLY.
*************************************************************************

INITACIA        MOVE.L  D4,-(A7)         ;PUSH
                MOVE.B  #3,ACIACS        ;RESET ACIA
                MOVE.W  #$1000,D4        ;TIMING CONSTANT
WAIT            DBRA    D4,WAIT          ;PAUSE
                MOVE.B  #ACIAMODE,ACIACS        ;SET ACIA MODE
                MOVE.L  #ACIAHANDLER,ACIAVECTOR ;TRAP THROUGH VECTOR 29
                MOVE.L  (A7)+,D4                ;POP
                RTS

*************************************************************************
****** THIS PROCEDURE FIGURES OUT WHAT KIND OF ACIA INTERRUPT HAS OCCURED
****** AND CALLS WHATEVER IT HAS TO TO HANDLE THE SITUATION.  THE THREE
****** POSSIBLE SITUATIONS ARE RECEIVE INTERRUPT (FROM THE SCREEN READY
****** FOR ANOTHER CHARACTER), TRANSMIT INTERRUPT (WHEN A KEY HAS BEEN
****** PRESSED ON THE KEYBOARD), OR OVERRUN.  IN PRACTICE THIS ROUTINE
****** NEVER CALLS THE OVERFLOW.
* A "-" WILL DIVIDE THE DELAY VALUE BY 2, A "+" WILL MULTIPLY IT BY 2
* A "#" WILL SIGNIFY THAT THE PROCESS IS NOW DEAD.  THE PROCESS WILL RESET
* THE TIMER SO THAT IT DOESN'T HAVE TO WAIT FOR ITS TIMESLICE TO EXPIRE.
* THUS, WHEN YOU KILL A PROCESS THE NEXT ONE IS SWAPPED IN WITHIN A FEW
* CYCLES.  I GIVE THE TIMER 25 CYCLES BEFORE THE NEXT INTERRUPT HITS SO
* THAT IT GIVES ACIAHANDLER A CHANCE TO FINISH EXCEPTION PROCESSING AND
* RETURN TO THE DELAY LOOP BEFORE THE NEXT PROCESS IS SWAPPED IN.
* THE REGISTERS A5, D5, AND D6 ARE DESTROYED AND NOT RESTORED.
*************************************************************************

ACIAHANDLER     BLOCKINT

*************************************************************************
* THE FIRST PART OF THE CODE FIGURES OUT FROM THE STATUS BITS OF THE
* ACIA WHAT KIND OF INTERRUPT HAS OCCURED, AND JUMPS ACCORDINGLY.
*************************************************************************

                BTST.B  #0,ACIACS               ;IS IT FROM THE KEYBOARD?
                BNE     RECEIVE                 ;IF SO, RECEIVE A KEY PRESS
                BTST.B  #1,ACIACS               ;IS THE SCREEN READY FOR MORE?
```

```
                BNE     SEND                    ;IF SO, BRANCH TO SEND
                JMP     ENDHANDLER              ;RETURN IF SPURIOUS INT.

*************************************************************************
* THIS CODE HANDLES AN INTERRUPT FROM THE KEYBOARD:  STRUCTURALLY, IT
* IS A WHOLE SERIES OF IF-THEN STATEMENTS.  IT IS WRITTEN AS ONE LONG
* ROUTINE RATHER THAN SUBROUTINES TO SPEED UP EXECUTION TIME AND TO
* PREVENT THE STACK FROM BEING FILLED WITH SUBROUTINE CALLS.
*************************************************************************

RECEIVE         MOVEA.L PCBPOINTER,A5           ;HOLD ADDRESS OF POINTER
                MOVE.B  ACIAD,D6                ;STORE IT IN D4
                CMP.B   #'+',D6                 ;IF (KEY='+') THEN T:=T*2
                BNE     NOT_MULTIPLY            ;ELSE
                CMP.B   #0,(A5)                 ;IS IT A NULL PROCESS?
                BEQ     ENDHANDLER              ;IF SO, DON'T MULTIPLY

* CALL THE MACRO TO DOUBLE THE USERS DELAY *

                MULTIPLY_T

                JMP     ENDHANDLER              ;CONTINUE WHILE LOOP
NOT_MULTIPLY    CMP.B   #'-',D6                 ;IF (KEY='-') THEN DIVIDE T/2
                BNE     NOT_DIVIDE              ;ELSE
                CMP.B   #0,(A5)                 ;IS IT A NULL PROCESS?
                BEQ     ENDHANDLER              ;IF SO, DON'T DIVIDE
                DIVIDE_T

* CALL THE MACRO DO CUT THE USERS DELAY TIME IN HALF *

                JMP     ENDHANDLER

*************************************************************
* THIS CODE HANDLES THE DEATH OF A PROCESS.  TO BE "KILLED"
* A PROCESS SIMPLY HAS ITS ID CHAR SET TO NULL AND PCOUNT REDUCED
* BY ONE, AND THEN FORCE A CONTEXT SWITCH IN 25 CYCLES.
* NOTE THAT IF THERE ARE NO PROCESSES REMAINING, WE DON'T
* WANT TO DO ANY MORE SWAPPING, SO WE LOAD THE STACK WITH THE
* ADDRESS OF THE WAIT_ON_P LOOP AND WHEN THE PROGRAM HITS AN
* RTE THE LOOP IS JUMPED BACK TO.
*************************************************************

NOT_DIVIDE      CMP.B   #'#',D6                 ;IS PROCESS KILLED?
                BNE     CALL_ADDP               ;OR CHAR IS SOMETHING ELSE
                CMP.B   #0,(A5)                 ;ONLY KILL A LIVE PROCESS
                BEQ     ENDHANDLER              ;NOT A DEAD ONE
```

```
                MOVEA.L PCBPOINTER,A5           ;ADDRESS POINTED TO
                MOVE.B  #0,(A5)                 ;MARK PROCESS FOR DEATH
                SUB.B   #1,PCOUNT               ;DECREASE NUMBER OF PROCESSES
                CMP.B   #0,PCOUNT               ;ARE THERE ANY LEFT?
                BNE     SWAP_IN_NEXT            ;IF SO, SWAP IN NEXT ONE
                MOVE.B  #0,TCR                  ;TURN OFF TIMER
                MOVE.W  (A7)+,D5                ;POP OFF SR
                MOVE.L  (A7)+,A5                ;POP OFF PC
                MOVE.L  #WAIT_ON_P,A5           ;LOAD WITH ADDRESS OF WAITLOOP
                MOVE.L  A5,-(A7)                ;PUSH ALTERED PC
                MOVE.W  A5,-(A7)                ;PUSH ALTERED SR
                JMP     ENDHANDLER              ;RETURN

SWAP_IN_NEXT    MOVE.B  #0,TCR                  ;TURN OFF TIMER
                MOVE.L  #25,D5                  ;NUMBER OF CYCLES
                MOVEA.L #CPR,A0                 ;COUNTER PRELOAD REGISTERS
                MOVEP.L D5,0(A0)                ;HAVE TO USE THIS INSTRUCTION
                MOVE.B  #$A1,TCR                ;RESTART TIMER
                JMP     ENDHANDLER              ;RETURN

*****************************************************************
* THIS CODE CALLS ADDPROCESS TO ADD A NEW PROCESS AND THEN FORCES
* A CONTEXT SWITCH WITHIN 25 CYCLES.  ADDPROCESS ITSELF TAKES CARE
* OF UPDATING THE PCOUNT IN CASE NO PROCESSES CAN BE ADDED.
*****************************************************************

CALL_ADDP       JSR     ADDPROCESS              ;ADD A PROCESS WITH KEY=ID
                MOVE.B  #0,TCR                  ;TURN OFF TIMER
                MOVE.L  #25,D5                  ;NUMBER OF CYCLES
                MOVEA.L #CPR,A0                 ;COUNTER PRELOAD REGISTERS
                MOVEP.L D5,0(A0)                ;HAVE TO USE THIS INSTRUCTION
                MOVE.B  #$A1,TCR                ;RESTART TIMER
ENDHANDLER      UNBLOCKINT
                RTE

*****************************************************************
* THIS CODE HANDLES AN INTERRUPT FROM THE SCREEN READY TO RECEIVE
* MORE INFORMATION.  IT CALLS DEQ TO REMOVE A BYTE OF TEXT FROM
* THE QUEUE AND THEN SENDS IT DIRECTLY TO THE ACIA DATA REGISTER.
*****************************************************************

SEND            JSR     DEQ                     ;RETURN BYTE IN D7
                MOVE.B  D7,ACIAD                ;SEND BYTE TO SCREEN
                UNBLOCKINT
                RTE
```

```
*************************************************************************
****** THIS SUBROUTINE INITIALIZES THE QUEUE WHICH WILL HOLD LENGTH BYTES.
****** NOTE THAT THE QUEUE MUST BE ONE BYTE BIGGER THAN THE MAXIMUM NUMBER
****** OF BYTES IT CAN HOLD, SO THAT YOU CAN TELL WHEN THE QUEUE IS FULL.
*************************************************************************

INITQ           MOVEA.L #QUEUE,A6       ;A6 POINTS TO THE QUEUE STRUCTURE
                MOVE.L  A6,-(A7)        ;PUSH POINTER
                MOVE.W  #0,(A6)         ;SET HEAD TO ZERO
                ADD.L   #2,A6           ;POINT TO TAIL OFFSET
                MOVE.W  #0,(A6)         ;THE TAIL=HEAD=0 FOR NEW EMPTY QUEUE
                MOVE.L  (A7)+,A6        ;POP THE QUEUE POINTER
                RTS                     ;RETURN

*************************************************************************
****** THIS SUBROUTINES PUTS A BYTE INTO THE QUEUE
****** FIRST, IT INCREMENTS TAIL AND MODS IT TO FIGURE OUT WHERE THE
****** BYTE SHOULD GO.  THEN, IT COMPARES THE HEAD AND TAIL TO SEE IF
****** THE QUEUE HAS OVERFLOWED OR NOT.  THEN, IT USES A BASE ADDRESS
****** AND THE OFFSET TO CALCULATE WHERE IN MEMORY TO PUT THE BYTE
****** AND FINALLY, THE BYTE GOES IN THE QUEUE!
****** THEN, THE RECIEVE INTERRUPT IS FROM THE ACIA1 IS TURNED ON SO
****** THE TERMINAL CAN REQUEST CHARACTERS TO BE SENT TO IT.
****** THE CONTENTS OF THE ADDRESS REGISTERS A1,A2,AND A3 ARE DESTROYED.
*************************************************************************

ENQ             MOVEA.L A6,A1           ;COPY POINTER
                MOD     (A1),A2         ;FIND REAL ADDRESS OF HEAD
                ADD.L   #2,A1           ;POINT TO TAIL OFFSET
                ADD.W   #1,(A1)         ;INCREMENT OFFSET
                MOD     (A1),A3         ;GET THE REAL ADDRESS OF TAIL
                CMPA.L  A2,A3           ;HAS HEAD CRASHED INTO TAIL?
                BEQ     SHUTDOWN        ;EXIT IF IT HAPPENS
                MOVE.B  D7,(A3)         ;PUT THE BYTE IN THE QUEUE
                MOVE.B  #$B5,ACIACS     ;TURN ON TRANSMIT (SCREEN) INTERRUPTS
                RTS                     ;RETURN TO MAIN LOOP

*************************************************************************
****** THIS SUBROUTINE PULLS A BYTE OUT OF THE QUEUE.  HEAD IS INCREMENTED
****** AND THE CHARACTER IS RETURNED AS THE PARAMETER.  IF THE QUEUE IS
****** THEN EMPTY, THE INTERRUPTS FROM THE TERMINAL ARE TURNED OFF
****** SO THAT IT DOES NOT REQUEST ANY MORE CHARACTERS.
****** MOD IS USED SO THAT THE INCREMENTS ARE WRAPPED AROUND.
****** THE CONTENTS OF THE ADDRESS REGISTERS A1, A2, AND A3 ARE DESTROYED.
*************************************************************************
```

```
DEQ             MOVEA.L A6,A1           ;COPY POINTER
                ADD.W   #1,(A1)         ;INCREMENT HEAD
                MOD     (A1),A2         ;CALCULATE THE HEAD ADDRESS TO USE
                MOVE.B  (A2),D7         ;GET THE BYTE FROM THE HEAD OF QUEUE
                ADD.L   #2,A1           ;POINT TO TAIL OFFSET
                MOD     (A1),A3         ;CALCULATE TAIL ADDRESS TO COMPARE
                CMPA.L  A2,A3           ;IS THE QUEUE EMPTY?  (HEAD=TAIL)
                BNE     QNOTEMPTY       ;IF SO, TURN OFF ACIA INTERRUPTS
                MOVE.B  #$95,ACIACS     ;TURN OFF TRANSMIT (SCREEN) INTERRUPTS
QNOTEMPTY       RTS                     ;RETURN

*****************************************************************
* THE MULTIPLEXER RUNS WHEN A TIMER INTERRUPT OCCURS.  THE STACK IS
* USED TO PLACE THE OLD SR AND SP FROM THE USER PROCESS WHICH WAS
* INTERRUPTED, AND THEN A7,D0,AND D1 ARE STORED.  NOTE THAT THE
* STACK POINTER IS STORED AS IT WAS WHEN THE INTERRUPT HANDLER
* BEGAN EXECUTING, SO THAT THE CONTEXT IS RESTORED PROPERLY BY
* THE RTE AT THE END OF THE HANDLER.  THE CIRCULAR DATA STRUCTURE
* IS WALKED THROUGH UNTIL THE NEXT ACTIVE PROCESS IS FOUND.
* NOTE THAT THIS MULTIPLEXER SHOULD NEVER BE CALLED IF THERE
* ARE NO ACTIVE PROCESSES, BECAUSE TIMER INTERRUPTS WOULD BE
* SHUT DOWN.
*****************************************************************

MULTIPLEXER     BLOCKINT
                MOVE.B          #$0,TCR         ;DISABLE TIMER
                MOVEA.L         PCBPOINTER,A3   ;LOAD ADDRESS HELD THERE
                MOVEA.L         A3,A5           ;COPY TO TEMPORARY POINTER
                ADD.L           #2,A3           ;MOVE TO STORAGE OF SR
                MOVE.W          (A7)+,(A3)      ;POP SR INTO PCB
                ADD.L           #2,A3           ;POINT TO PC
                MOVE.L          (A7)+,(A3)      ;POP OLD PC IN
                ADD.L           #4,A3           ;POINT TO STORAGE OF D0
                MOVE.L          D0,(A3)         ;COPY D0 INTO PCB
                ADD.L           #4,A3           ;POINT TO STORAGE OF D1
                MOVE.L          D1,(A3)         ;COPY D1 INTO PCB
                ADD.L           #4,A3           ;POINT TO STORAGE OF D2
                MOVE.L          D2,(A3)         ;COPY D2 INTO PCB
                ADD.L           #4,A3           ;POINT TO NEXT PCB

*****************************************************************************
* THIS PORTION OF THE CODE WALKS THROUGH THE PCBS TO FIND THE NEXT
* ONE (CIRCULARLY) WHICH IS ACTIVE AND RESTORES IT.
*****************************************************************************

LOOK            MOVEA.L         #PCBPOINTER,A4  ;COPY POINTER'S ADDRESS
```

```
                CMPA.L          A3,A4           ;HAVE WE HIT BOTTOM?
                BNE             IS_IT_ACTIVE    ;IF NOT, TEST FOR ACTIVE
                MOVEA.L         #PCBTOP,A3      ;IF SO, WRAPAROUND
IS_IT_ACTIVE    CMP.B           #0,(A3)         ;IF NOT,
                BEQ             KEEP_LOOKING    ;KEEP LOOKING
                JMP             RESTORECONTEXT  ;OR ELSE SWAP IT IN
KEEP_LOOKING    ADD.L           #20,A3          ;POINT TO NEXT PROCESS
                JMP             LOOK            ;LOOP AROUND

*************************************************************
* THIS CODE RESTORES THE CONTEXT OF A PROCESS FROM THE PCB
*************************************************************

RESTORECONTEXT  MOVE.L          A3,PCBPOINTER   ;UPDATE POINTER
                ADD.L           #4,A3           ;POINT TO STORAGE OF PC
                MOVE.L          (A3),-(A7)      ;PUSH OLD PC
                SUB.L           #2,A3           ;POINT TO STORED SR
                MOVE.W          (A3),-(A7)      ;PUSH OLD SR
                ADD.L           #6,A3           ;POINT TO STORAGE OF D0
                MOVE.L          (A3),D0         ;RESTORE D0
                ADD.L           #4,A3           ;POINT TO STORAGE OF D1
                MOVE.L          (A3),D1         ;RESTORE D1
                ADD.L           #4,A3           ;POINT TO STORAGE OF D2
                MOVE.L          (A3),D2         ;RESTORE D2

*************************************************************************
* THIS CODE FINISHES UP AND RETURNS CONTROL TO THE NEXT USER PROCESS
*************************************************************************

CONTINUE        MOVE.L  TIMESLICE,D4    ;NUMBER OF CYCLES
                MOVEA.L #CPR,A5         ;COUNTER PRELOAD REGISTERS
                MOVEP.L D4,0(A5)        ;HAVE TO USE THIS INSTRUCTION
                MOVE.B  #$A1,TCR        ;ENABLE TIMER
                UNBLOCKINT
                RTE                     ;CONTINUE EXECUTION OF USER PROCESS

*************************************************************************
************** ADD PROCESS ADDS A PROCESS TO THE LIST ***************
* THIS SUBROUTINE IS CALLED WHEN A KEY BESIDES THE +,-, OR # IS PRESSED.
* IT WALKS THROUGH THE LIST OF PCBS UNTIL IT FINDS A NULL PROCESS OR
* HAS WALKED ALL THE WAY AROUND WITHOUT FINDING ONE.  IF IT FINDS A
* NULL PROCESS THE CHARACTER OF THE KEY PRESSED IS MADE THE ID CHAR OF
* A NEW PROCESS AND THE PROCESS IS INSERTED INTO THE PCB LIST.
*************************************************************************

ADDPROCESS      MOVEA.L PCBPOINTER,A3           ;LOAD ADDRESS
```

```
                MOVEA.L #PCBPOINTER,A4          ;MEMORY LOCATION
                MOVEA.L A3,A5                   ;COPY TO TEMPORARY POINTER
WRAP            CMPA.L  A3,A4                   ;HAVE WE HIT BOTTOM
                BNE     IS_IT_USED              ;IF NOT CHECK FOR ALL FULL
                MOVEA.L #PCBTOP,A3              ;OR ELSE WRAPAROUND
IS_IT_USED      CMP.B   #0,(A3)                 ;IS PROCESS USED?
                BEQ     ADDIT                   ;IF NOT, ADD HERE
ARE_ALL_USED    ADD.L   #20,A3                  ;POINT TO NEXT PCB
                CMPA.L  A3,A5                   ;HAVE WE GONE AROUND?
                BNE     PCB_LOOP                ;IF NOT, CONTINUE
                RTS                             ;OR QUIT WITHOUT ADDING
PCB_LOOP        JMP     WRAP                    ;CONTINUE SEARCH
ADDIT           MOVE.B  D6,(A3)                 ;SET PROCESS ID
                ADD.B   #1,PCOUNT               ;PCOUNT:=PCOUNT+1
                ADD.L   #8,A3                   ;POINT TO DELAY IN PCB
                MOVE.L  #DELAY1,(A3)            ;STORE INITIAL DELAY=1 SEC.
                CMP.B   #1,PCOUNT               ;IS THIS THE FIRST PROCESS
                BNE     DOWNHERE                ;IF NOT, DON'T SET D0
                MOVE.L  #DELAY1,D0              ;SET UP D0 FOR FIRST DELAY
DOWNHERE        RTS                             ;RETURN TO ACIAHANDLER

*************************************************************************
* MAIN PROGRAM
* THIS FIRST SETS UP THE PCBPOINTER, THEN THE QUEUE, THEN THE ACIA,
* THEN THE TIMER, THEN UNBLOCKS THE INTERRUPTS AND ENTERS
* THE LOOP TO WAIT FOR THE FIRST PROCESS TO BE INVOKED BY ADDPROCESS.
*************************************************************************

START           MOVEA.L #$7000,A7       ;SET STACK POINTER TO $7000
                JSR     INITQ           ;SET UP THE I/O QUEUE
                JSR     INITPCBS        ;SET UP THE 16 PCBS IN RAM
                JSR     INITALINE       ;SET UP THE A-LINE EMULATOR
                JSR     TIMERINIT       ;START TIMER INTERRUPTS
                JSR     INITACIA        ;SET UP THE ACIA

                MOVE.B  #$95,ACIACS     ;TURN ON RECEIVE (KEY) INTERRUPTS
                UNBLOCKINT

*************************************************************************

WAIT_ON_P       CMP.B   #0,PCOUNT       ;ARE THERE NO PROCESSES?
                BEQ     WAIT_ON_P       ;IF SO, LOOP HERE

*************************************************************************
* THIS IS THE SHARED CODE:  IT CONSISTS OF THE DISPLAY MACRO (WHICH IS
* A SYSTEM CALL) AND THE CODE FOR THE DELAY PROCEDURE.  THE REGISTERS
```

```
* USED BY THE MAIN PROCEDURE ARE D0, D1 AND D2 FOR THE CONSTANT DELAY,
* OUTER LOOP DELAY TIMER, AND UNITDELAY TIMER.  THESE REGISTERS ARE
* STORED WHEN THE PROCESS IS SWAPPED IN AND OUT.
************************************************************************

WHILELOOP       DISPLAY                 ;CALL THE A-LINE HANDLER TO DISPLAY

************************************************************************
* THIS IS THE CODE FOR DELAY.  I AM LEAVING IT IN THE MAIN PROCEDURE
* SO THAT THE CONTEXT SWITCHER WON'T HAVE SUBROUTINE CALLS ON THE
* STACK TO CONTEND WITH, AND SO EACH PROCESS CAN RESUME PROPERLY.
* SEPARATE STACKS WOULD BE EASIER WITH A MEMORY MANAGEMENT UNIT.
************************************************************************

                MOVE.L  D0,D1           ;COPY T TO LOCAL
DELOOP          UNITDELAY               ;CALL UNITDELAY
                DBEQ    D1,DELOOP       ;REPEAT T TIMES
                JMP     WHILELOOP       ;GO BACK TO DISPLAY: REPEAT

************************************************************************
* SHUTDOWN IS CALLED WHEN THE QUEUE OVERFLOWS.
************************************************************************

SHUTDOWN        MOVE.B  #$0,TCR         ;DISABLE TIMER
                MOVE.B  #229,D7         ;RETURN CONTROL
                TRAP    #14             ;TO TUTOR
                END     TOP             ;END OF THE PROGRAM
```