*File created:* 29-Jul-2023 11:47:41 **{DSK}<home>larry>il>medley>sources>PROC.;7**

*edit by:* lmm

*changes to:* (VARS PROCCOMS)
(FNS \BACKGROUND.PROCESS)

*previous date:* 28-Jul-2023 21:06:03 {DSK}<home>larry>il>medley>sources>PROC.;6

*Read Table:* INTERLISP

*Package:* INTERLISP

*Format:* XCCS

```
(RPAQQ PROCCOMS
       [(COMS (DECLARE%: DONTCOPY (EXPORT (RECORDS PROCESS))
                      (RECORDS PROCESSQUEUE)
                      (CONSTANTS \PSTAT.WAITING \PSTAT.RUNNING \PSTAT.DELETED))
              (INITRECORDS PROCESS PROCESSQUEUE)
              (SYSRECORDS PROCESS PROCESSQUEUE))
        [COMS                                                    ; User entries
              (FNS PROCESSWORLD ADD.PROCESS DEL.PROCESS PROCESS.RETURN FIND.PROCESS MAP.PROCESSES PROCESSP
                   RELPROCESSP RESTART.PROCESS WAKE.PROCESS SUSPEND.PROCESS PROCESS.RESULT PROCESS-STATUS
                   PROCESS.FINISHEDP)
              (FNS THIS.PROCESS TTY.PROCESS TTY.PROCESSP PROCESS.TTY GIVE.TTY.PROCESS ALLOW.BUTTON.EVENTS
                   SPAWN.MOUSE \WAIT.FOR.TTY WAIT.FOR.TTY)
              (FNS RESET ERROR!)
              [INITVARS (TTY.PROCESS.DEFAULT '(EXEC MOUSE))
                      (\TTY.PROCESS.EVENT)
                      (\TTY.PROCESS)
                      (\PROCESS.NAME.TABLE (HASHARRAY 30 NIL (FUNCTION STRING-EQUAL-HASHBITS)
                                                 (FUNCTION STRING-EQUAL]
              (GLOBALVARS TTY.PROCESS.DEFAULT \TTY.PROCESS.EVENT \PROCESS.NAME.TABLE)
              (FNS PROCESSPROP PROCESS.NAME PROCESS.WINDOW)
              (PROP ARGNAMES PROCESSPROP ADD.PROCESS)
              (COMS                                              ; Temporary
                    (P (MOVD? 'PROCESS.RETURN 'KILL.ME NIL T]
        (COMS (FNS DISMISS BLOCK WAITFORINPUT \WAITFORSYSBUFP)
                                                                 ; Used to be a GLOBALRESOURCES
              (DECLARE%: DONTCOPY (RESOURCES \DISMISSTIMER))
              (INITRESOURCES \DISMISSTIMER))
        (COMS (FNS EVAL.AS.PROCESS EVAL.IN.TTY.PROCESS)
              ;; The PROCESS.WAIT macro is an augmentation to BLOCK, waiting for a condition to come true, or a timeout, or a wakeup
              (MACROS PROCESS.WAIT)
              (FNS PROCESS.READ PROCESS.EVALV PROCESS.EVAL \PROCESS.EVAL1 PROCESS.APPLY \PROCESS.APPLY1)
                                                                 ; Standard values for WAKEREASON -- PSTAT.TIMEDOUT is
                                                                 ; the only public one
              (VARS (PSTAT.WAKEUP "default WakeUp")
                    (PSTAT.TIMEDOUT "{time interval expired}")
                    (PSTAT.QUIT "Quit")
                    (\PSTAT.NORESULT "{no result yet}"))
              (GLOBALVARS PSTAT.WAKEUP PSTAT.TIMEDOUT PSTAT.QUIT \PSTAT.NORESULT))
        (COMS                                                    ; Event stuff
              (DECLARE%: DONTCOPY (RECORDS EVENT))
              (INITRECORDS EVENT)
              (SYSRECORDS EVENT)
              (FNS CREATE.EVENT NOTIFY.EVENT AWAIT.EVENT \UNQUEUE.EVENT \ENQUEUE.EVENT/LOCK \EVENT.DEFPRINT)
              (MACROS AWAIT.CONDITION)
              (INITVARS (\PROCESS.AFTEREXIT.EVENT))
              (GLOBALVARS \PROCESS.AFTEREXIT.EVENT))
        (COMS                                                    ; Monitor stuff
              (DECLARE%: DONTCOPY (RECORDS MONITORLOCK)
                     (MACROS .RELEASE.LOCK.))
              (INITRECORDS MONITORLOCK)
              (SYSRECORDS MONITORLOCK)
              (FNS OBTAIN.MONITORLOCK CREATE.MONITORLOCK RELEASE.MONITORLOCK SI::MONITOR-UNWIND
                   MONITOR.AWAIT.EVENT \MONITORLOCK.DEFPRINT)
              (MACROS WITH.MONITOR WITH.FAST.MONITOR))
        (COMS (EXPORT (SPECVARS \BACKGROUND)
                      (GLOBALVARS \IGNORE.BACKGROUND))
              (INITVARS (\BACKGROUND NIL)
                      (\IGNORE.BACKGROUND T))
              (FNS \MAKE.PROCESS0 \MAKE.PROCESS1 \PROCESS.MOVEFRAME \RELEASE.PROCESS \UNWIND.PROCESS \MAYBEBLOCK
                   \BACKGROUND.PROCESS \MOUSE.PROCESS \TIMER.PROCESS \PROCESS.RELEASE.LOCKS \SET.PROCESS.NAME
                   \PROCESS.DEFPRINT)
              (FNS \START.PROCESSES \PROCESS.GO.TO.SLEEP \PROC.RESUME \RUN.PROCESS \SUSPEND.PROCESS
                   \UNQUEUE.TIMER \ENQUEUE.TIMER \GET.PRIORITY.QUEUE)
              (DECLARE%: DONTCOPY (MACROS \RESCHEDULE)))
        (COMS (FNS \PROCESS.INIT \PROCESS.EVENTFN \PROCESS.BEFORE.LOGOUT \PROCESS.AFTER.EXIT
                   \PROCESS.RESET.TIMERS \PROC.AFTER.WINDOWWORLD \TURN.ON.PROCESSES)
                                                                 ; Redefinitions
              (FNS \PROC.CODEFORTFRAME \PROC.REPEATEDLYEVALQT))
        (COMS                                                    ; switching stacks
              (FNS BREAK.PROCESS \SELECTPROCESS \PROCESS.MAKEFRAME \PROCESS.MAKEFRAME0))
        (INITVARS (%#MYHANDLE#)
                  (%#SCHEDULER#)
```

```
               (\RUNNING.PROCESS)
               (\PROCESSES)
               (PROCESS.MAXMOUSE 5)
               (PROC.FREESPACESIZE 1024)
               (AUTOPROCESSFLG T)
               (BACKGROUNDFNS)
               (\TIMERQHEAD)
               (\HIGHEST.PRIORITY.QUEUE)
               (PROC.DEFAULT.PRIORITY 2)
               (\DEFAULTLINEBUF)
               (\DEFAULTTTYDISPLAYSTREAM)
               (\PROCTIMER.SCRATCH (NCREATE 'FIXP))
               (TOPW)
               (\PROC.RUN.NEXT.FLG)
               (\PROC.READY T))
        (ADDVARS (\SYSTEMCACHEVARS \PROC.READY)
               (\SYSTEMTIMERVARS (\LASTUSERACTION SECONDS)))
        (COMS (VARS (\PROC.RESTARTME "{restart flag}")
                    (\PROC.RESETME "{reset flag}")
                    (\PROC.KILLME "{abort flag}"))
              (DECLARE%: DONTCOPY (EXPORT (MACROS THIS.PROCESS TTY.PROCESS TTY.PROCESSP)
                                          (GLOBALVARS \RUNNING.PROCESS \TTY.PROCESS \PROC.RESTARTME \PROC.RESETME
                                                      \PROC.ABORTME))
                     (GLOBALVARS \PROCESSES PROC.FREESPACESIZE %#SCHEDULER# PROCESS.MAXMOUSE AUTOPROCESSFLG
                            BACKGROUNDFNS \TopLevelTtyWindow \PROC.READY)
                     (GLOBALVARS \TIMERQHEAD \PROCTIMER.SCRATCH \HIGHEST.PRIORITY.QUEUE PROC.DEFAULT.PRIORITY
                            \PROC.RUN.NEXT.FLG \SYSTEMTIMERVARS)
                     (MACROS ALIVEPROCP DEADPROCP \COERCE.TO.PROCESS)
                     (LOCALVARS . T)))
        (COMS                                                    ; Debugging
              (FNS \CHECK.PQUEUE)
              (FNS PPROC PPROCWINDOW PPROCREPAINTFN PPROCRESHAPEFN PPROCEXTENT PPROC1 PROCESS.STATUS.WINDOW
                   \PSW.SELECTED \PSWOP.SELECTED PROCESS.BACKTRACE \INVALIDATE.PROCESS.WINDOW
                   \UPDATE.PROCESS.WINDOW)
              (INITVARS (PROCMENU)
                     (PROCOPMENU)
                     (PROCOP.WAKEMENU)
                     (PROCESS.STATUS.WINDOW)
                     (SELECTEDPROC)
                     (PROCBACKTRACEHEIGHT 320))
              (ADDVARS (BackgroundMenuCommands ("PSW" '(PROCESS.STATUS.WINDOW)
                                                "Puts up a Process Status Window")))
              (P (SETQQ BackgroundMenu))
              (DECLARE%: EVAL@COMPILE DONTCOPY (GLOBALVARS PROCESS.STATUS.WINDOW PROCMENU PROCOPMENU
                                                PROCOP.WAKEMENU PROCBACKTRACEHEIGHT SELECTEDPROC
                                                BACKTRACEFONT)
                     (CONSTANTS LIGHTGRAYSHADE SELECTIONSHADE)))
        (DECLARE%: DONTEVAL@LOAD DOCOPY (ADDVARS (WINDOWUSERFORMS (\PROC.AFTER.WINDOWWORLD)))
               (P (DEFPRINT 'PROCESS (FUNCTION \PROCESS.DEFPRINT))
                  (DEFPRINT 'EVENT (FUNCTION \EVENT.DEFPRINT))
                  (DEFPRINT 'MONITORLOCK (FUNCTION \MONITORLOCK.DEFPRINT))
                                                             ; \process.init must come last, since it does a HARDRESET
                  (\PROCESS.INIT)))
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                              (NLAML)
                                                                              (LAMA PROCESSPROP ADD.PROCESS])

(DECLARE%: DONTCOPY

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: EVAL@COMPILE

(DATATYPE PROCESS ((PROCFX0 WORD)                            ; = \STACKHI to make this look like a STACKP
                   (PROCFX WORD)                             ; Stack pointer to this context when it is asleep
                   (PROCSTATUS BYTE)                         ; Running or waiting
                   (PROCNAME POINTER)                        ; Name for convenience in type-in reference
                   (PROCPRIORITY BYTE)                       ; Priority level, 0-4.  Not currently used.
                   (PROCQUEUE POINTER)                       ; Queue of processes at the same priority
                   (NIL BYTE)
                   (NEXTPROCHANDLE POINTER)                  ; Pointer to next one
                   (PROCTIMERSET FLAG)                       ; True if PROCWAKEUPTIMER has an interesting value
                   (PROCBEINGDELETED FLAG)                   ; True if proc was deleted, but hasn't been removed from
                                                             ; \PROCESSES yet
                   (PROCDELETED FLAG)
                   (PROCSYSTEMP FLAG)
                   (PROCNEVERSTARTED FLAG)
                   (NIL FLAG)
                   (NIL FLAG)
                   (NIL FLAG)
                   (PROCWAKEUPTIMER POINTER)                 ; a largep recording the time this proc last went to sleep
                   (PROCTIMERLINK POINTER)                   ; For linking proc in timer queue
                   (PROCTIMERBOX POINTER)                    ; Scratch box to use for PROCWAKEUPTIMER when user does
                                                             ; not give one explicitly
                   (WAKEREASON POINTER)                      ; Reason process is being run.  From WAKE.PROCESS or timer
                                                             ; or event wakeup;  T from simple BLOCK
                   (PROCEVENTORLOCK POINTER)                 ; EVENT or MONITOR lock that this proc is waiting for
```

```
                    (PROCFORM POINTER)                          ; Form to EVAL to start it going
                    (RESTARTABLE POINTER)                       ; T = autorestart on error, HARDRESET = restart only on hard
                                                                ; reset, NIL = never restart
                    (PROCWINDOW POINTER)                        ; Window this process lives in, if any
                    (PROCFINISHED POINTER)                      ; True if proc finished.  Value is indication of how: NORMAL,
                                                                ; DELETED, ERROR
                    (PROCRESULT POINTER)                        ; Value it returned if it finished normally
                    (PROCFINISHEVENT POINTER)                   ; Optional EVENT to be notified when proc finishes
                    (PROCMAILBOX POINTER)                       ; Message queue
                    (PROCDRIBBLEOUTPUT POINTER)                 ; Binding for *DRIBBLE-OUTPUT* in this process
                    (PROCINFOHOOK POINTER)                      ; Optional user fn that displays info about process
                    (PROCTYPEAHEAD POINTER)                     ; Buffer of typeahead destined for this proc
                    (PROCREMOTEINFO POINTER)                    ; For Enterprise
                    (PROCUSERDATA POINTER)                      ; For PROCESSPROP
                    (PROCEVENTLINK POINTER)                     ; Used to maintain EVENT queues
                    (PROCAFTEREXIT POINTER)                     ; What to do with this process when coming back from a
                                                                ; LOGOUT, etc
                    (PROCBEFOREEXIT POINTER)                    ; If DON'T, can't logout
                    (PROCOWNEDLOCKS POINTER)                    ; Pointer to first lock I currently own
                    (PROCEVAPPLYRESULT POINTER)                 ; For PROCESS.EVAL and PROCESS.APPLY when
                                                                ; WAITFORRESULT is true
                    (PROCTTYENTRYFN POINTER)                    ; Is applied to a process when it becomes the tty process
                    (PROCTTYEXITFN POINTER)                     ; Is applied to a process when it ceases to be the tty process
                    (PROCHARDRESETINFO POINTER)                 ; HARDRESET stores info about unwind-protect cleanups here
                    (PROCRESTARTFORM POINTER)                   ; use this instead of PROCFORM when restarting
                    (PROCOLDTTYPROC POINTER)                    ; Process that had the tty when we got it
                    (NIL POINTER)                               ; For expansion

                   )
       PROCTIMERBOX _ (CREATECELL \FIXP)
       PROCFX0 _ \STACKHI)
)

(/DECLAREDATATYPE 'PROCESS
       '(WORD WORD BYTE POINTER BYTE POINTER BYTE POINTER FLAG FLAG FLAG FLAG FLAG FLAG FLAG FLAG POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '66)

;; END EXPORTED DEFINITIONS

(DECLARE%: EVAL@COMPILE

(DATATYPE PROCESSQUEUE ((PQPRIORITY BYTE)                       ; Priority for the processes in thie queue.
                        (PQHIGHER POINTER)                      ; Next higher-prioirty queue
                        (PQLOWER POINTER)                       ; Next lower
                        (PQNEXT POINTER)                        ; The process currently running or runnable at this priority
                        (PQLAST POINTER)                        ; The proc previous to it.  PQNEXT might be redundant

                        ))
)

(/DECLAREDATATYPE 'PROCESSQUEUE '(BYTE POINTER POINTER POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '10)

(DECLARE%: EVAL@COMPILE

(RPAQQ \PSTAT.WAITING 0)

(RPAQQ \PSTAT.RUNNING 1)

(RPAQQ \PSTAT.DELETED 2)

(CONSTANTS \PSTAT.WAITING \PSTAT.RUNNING \PSTAT.DELETED)
)
)

(/DECLAREDATATYPE 'PROCESS
       '(WORD WORD BYTE POINTER BYTE POINTER BYTE POINTER FLAG FLAG FLAG FLAG FLAG FLAG FLAG FLAG POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '66)

(/DECLAREDATATYPE 'PROCESSQUEUE '(BYTE POINTER POINTER POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '10)

(ADDTOVAR SYSTEMRECLST
```

```
          (DATATYPE PROCESS ((PROCFX0 WORD)
                             (PROCFX WORD)
                             (PROCSTATUS BYTE)
                             (PROCNAME POINTER)
                             (PROCPRIORITY BYTE)
                             (PROCQUEUE POINTER)
                             (NIL BYTE)
                             (NEXTPROCHANDLE POINTER)
                             (PROCTIMERSET FLAG)
                             (PROCBEINGDELETED FLAG)
                             (PROCDELETED FLAG)
                             (PROCSYSTEMP FLAG)
                             (PROCNEVERSTARTED FLAG)
                             (NIL FLAG)
                             (NIL FLAG)
                             (NIL FLAG)
                             (PROCWAKEUPTIMER POINTER)
                             (PROCTIMERLINK POINTER)
                             (PROCTIMERBOX POINTER)
                             (WAKEREASON POINTER)
                             (PROCEVENTORLOCK POINTER)
                             (PROCFORM POINTER)
                             (RESTARTABLE POINTER)
                             (PROCWINDOW POINTER)
                             (PROCFINISHED POINTER)
                             (PROCRESULT POINTER)
                             (PROCFINISHEVENT POINTER)
                             (PROCMAILBOX POINTER)
                             (PROCDRIBBLEOUTPUT POINTER)
                             (PROCINFOHOOK POINTER)
                             (PROCTYPEAHEAD POINTER)
                             (PROCREMOTEINFO POINTER)
                             (PROCUSERDATA POINTER)
                             (PROCEVENTLINK POINTER)
                             (PROCAFTEREXIT POINTER)
                             (PROCBEFOREEXIT POINTER)
                             (PROCOWNEDLOCKS POINTER)
                             (PROCEVAPPLYRESULT POINTER)
                             (PROCTTYENTRYFN POINTER)
                             (PROCTTYEXITFN POINTER)
                             (PROCHARDRESETINFO POINTER)
                             (PROCRESTARTFORM POINTER)
                             (PROCOLDTTYPROC POINTER)
                             (NIL POINTER)))
          (DATATYPE PROCESSQUEUE ((PQPRIORITY BYTE)
                                  (PQHIGHER POINTER)
                                  (PQLOWER POINTER)
                                  (PQNEXT POINTER)
                                  (PQLAST POINTER))))
```

;; User entries

```
(DEFINEQ
```

## (**PROCESSWORLD**

```
  [LAMBDA (FLG)                                          ; Edited  1-Jun-88 15:39 by bvm
```

;; get started with multi-processing

```
    (COND
       [(EQ FLG 'OFF)                                    ; Turn them off
```

;; Release the stack space used by the procs, but keep the handles around for possible unwinding; normally processworld is never turned
;; off--don't know if this has any hope of working any more.

```
        (for P in \PROCESSES do (\RELEASE.PROCESS P))
        (SETQ \TTY.PROCESS)
        (COND
           ((TYPENAMEP \TopLevelTtyWindow 'WINDOW)
            (WINDOWPROP \TopLevelTtyWindow 'PROCESS NIL)))
        (SETQ \RUNNING.PROCESS)
        (COND
           ((AND %#SCHEDULER# (NEQ 0 (fetch PROCFX of %#SCHEDULER#)))
            (RETTO (PROG1 %#SCHEDULER# (SETQ %#SCHEDULER#))
                   PSTAT.QUIT T]
       (\RUNNING.PROCESS "Processes are already on")
       (T (PROG ((FIRSTTIME (NOT (type? PROCESS %#SCHEDULER#)))
                 EXECPROC BACKGROUNDPROC)
                [PROGN (SETQ \STACKOVERFLOW NIL)          ; Clear the stack overflow indicator in case a hard reset occurred.
                       (COND
                          (\WINDOWWORLD                   ; Cursor maybe smashed if died in hard stack overflow.  Only do
                                                          ; this if window world on (bootstrap problem).
                                 (CURSOR T]
                (COND
                   (FIRSTTIME (SETQ %#SCHEDULER# (create PROCESS))
                              (SETQ \TTY.PROCESS.EVENT (CREATE.EVENT 'TTY))
                              (SETQ \PROCESS.AFTEREXIT.EVENT (CREATE.EVENT "After Exit")))
                   (T (replace PROCFX of %#SCHEDULER# with 0)))
```

```
;; First wander thru any old processes, checking for unwind info and processes that said they want to restart on HARDRESET
      [COND
         ((type? PROCESSQUEUE \HIGHEST.PRIORITY.QUEUE) ; Empty out the queues
          (for (PQ _ \HIGHEST.PRIORITY.QUEUE) by (fetch PQLOWER of PQ) while PQ
             do (replace PQNEXT of PQ with (replace PQLAST of PQ with NIL]
      (SETQ \PROCESSES (for P in \PROCESSES when (COND
                                                    ((EQ (fetch PROCNAME of P)
                                                         'EXEC)
                                                        ; Save the primary EXEC to run last
                                                     (\RELEASE.PROCESS P)
                                                     (SETQ EXECPROC P)
                                                     NIL)
                                                    ((fetch PROCNEVERSTARTED of P)
                                                        ; Process got created when scheduling was off
                                                     (replace PROCNEVERSTARTED of P with NIL)
                                                     T)
                                                    ((fetch RESTARTABLE of P)
                                                        ; Stack of this process got flushed by a hard reset
                                                     T)
                                                    ((OR (AND (EQ P \TTY.PROCESS)
                                                              (fetch PROCTTYEXITFN of P))
                                                         (fetch PROCDRIBBLEOUTPUT of P)
                                                         (fetch PROCHARDRESETINFO of P))
                                                        ; Need to clean up once processworld back on
                                                     (replace PROCFINISHED of P with 'DELETED)
                                                     T)
                                                    (T  ; Not restartable & no cleanup, so just bash it.
                                                        (replace PROCDELETED of P with T)
                                                        (\RELEASE.PROCESS P T T)
                                                        ; 3rd arg tells it not to remove it from \processes, because we're
                                                        ; doing that.
                                                        NIL))
                                collect (PROGN (\RELEASE.PROCESS P)
                                                        ; Take it off any queues etc it was on
                                               P)))
      (for P in \PROCESSES do                           ; Bring it back to life
                                (\MAKE.PROCESS0 (OR (fetch PROCRESTARTFORM of P)
                                                    (fetch PROCFORM of P))
                                       P)
                                (\RUN.PROCESS P))
      [COND
         ([NOT (SETQ BACKGROUNDPROC (FIND.PROCESS 'BACKGROUND]
          (SETQ BACKGROUNDPROC (ADD.PROCESS (LIST (FUNCTION \BACKGROUND.PROCESS))
                                            'NAME
                                            'BACKGROUND
                                            'RESTARTABLE
                                            'SYSTEM
                                            'SCHEDULE T]
      (COND
         ((NOT (FIND.PROCESS 'MOUSE))
          (ADD.PROCESS (LIST (FUNCTION \MOUSE.PROCESS))
                 'NAME
                 'MOUSE
                 'RESTARTABLE
                 'SYSTEM
                 'SCHEDULE T)))
      (COND
         ((NOT (FIND.PROCESS '\TIMER.PROCESS))
          (SETQ \TIMERQHEAD (ADD.PROCESS (LIST (FUNCTION \TIMER.PROCESS))
                                         'RESTARTABLE
                                         'SYSTEM
                                         'SCHEDULE T)))
         (T (replace PROCTIMERLINK of (\DTEST \TIMERQHEAD 'PROCESS) with NIL)))
      [COND
         (EXECPROC                                      ; Restore exec last so that it at least starts out with all of stack
                                                        ; space to play with, don't sandbar as soon
                 (push \PROCESSES EXECPROC)
                 (\MAKE.PROCESS0 (fetch PROCFORM of EXECPROC)
                        EXECPROC)
                 (\RUN.PROCESS EXECPROC))
         (FIRSTTIME                                     ; Create an exec.  Don't do this on Hard reset--if user has
                                                        ; deliberately killed exec, don't bring it back
                 (SETQ EXECPROC (ADD.PROCESS '(\PROC.REPEATEDLYEVALQT)
                                             'NAME
                                             'EXEC
                                             'RESTARTABLE
                                             'ALWAYS
                                             'SCHEDULE T]
      [COND
         ((NOT (MEMB \TTY.PROCESS \PROCESSES))          ; The tty process died in the hardreset, so make it the exec, or
                                                        ; background if no exec
          (SETQ \TTY.PROCESS (OR EXECPROC BACKGROUNDPROC]
;; All set to go now -- schedule a process, save state of this piece of stack in #Scheduler#.  Should never need to go around this loop,
;; says here.
 LP   [PROGN (replace NEXTPROCHANDLE of %#SCHEDULER# with (CAR \PROCESSES))
             (LET ((RESULT (\START.PROCESSES)))
```

```
                        (COND
                           ((EQ RESULT PSTAT.QUIT)          ; from (PROCESSWORLD 'OFF)
                            (RETFROM 'PROCESSWORLD))
                           (T (RAID "??? Process error - strange result from \Start.Processes" RESULT]
              (GO LP])
```

## (ADD.PROCESS

```
  [LAMBDA ARGS                                              ; Edited 8-May-87 17:36 by bvm
    (PROG ((CREATENOW (THIS.PROCESS))
           (PRIORITY PROC.DEFAULT.PRIORITY)
           FORM RESTARTFLG SYSTEMP SUSPENDIT INFOHOOK RESTARTFORM WINDOW NAME AFTEREXIT PROC USERPROPS PROP
           VALUE BEFOREEXIT TTYENTRYFN TTYEXITFN)
          [COND
             ([OR (EQ ARGS 0)
                  (NLISTP (SETQ FORM (ARG ARGS 1]
              (RETURN (\ILLEGAL.ARG FORM]
          [COND
             ((EQ ARGS 2)                                   ; Backward compatibility
              (SETQ NAME (ARG ARGS 2)))
             (T (for I from 2 to ARGS by 2 do (SETQ VALUE (ARG ARGS (ADD1 I)))
                                              (SELECTQ (SETQ PROP (ARG ARGS I))
                                                  (WINDOW (SETQ WINDOW (\INSUREWINDOW VALUE)))
                                                  (PRIORITY (SETQ PRIORITY (\DTEST VALUE 'SMALLP)))
                                                  (NAME (SETQ NAME VALUE))
                                                  (AFTEREXIT (SETQ AFTEREXIT VALUE))
                                                  (BEFOREEXIT (SETQ BEFOREEXIT VALUE))
                                                  (TTYENTRYFN (SETQ TTYENTRYFN VALUE))
                                                  (TTYEXITFN (SETQ TTYEXITFN VALUE))
                                                  (INFOHOOK (SETQ INFOHOOK VALUE))
                                                  (RESTARTFORM (SETQ RESTARTFORM VALUE))
                                                  (RESTARTABLE (SETQ RESTARTFLG VALUE))
                                                  (SCHEDULE (SETQ CREATENOW T))
                                                  (SUSPEND (SETQ SUSPENDIT VALUE))
                                                  (COND
                                                     ([AND (EQ ARGS 3)
                                                           (FMEMB VALUE '(SYSTEM NO T]
                                                           ; Backward compatibility: arglist used to be (FORM NAME
                                                           ; RESTARTFLG)
                                                      (SETQ NAME PROP)
                                                      (SETQ RESTARTFLG VALUE))
                                                     (T (push USERPROPS PROP VALUE]
          (SETQ RESTARTFLG (SELECTQ RESTARTFLG
                                (SYSTEM (SETQ SYSTEMP T))
                                ((NIL NO NEVER)
                                    NIL)
                                ((T YES ALWAYS)
                                    T)
                                (HARDRESET 'HARDRESET)
                                (\ILLEGAL.ARG RESTARTFLG)))
          (SETQ PROC
           (create PROCESS
                   PROCTIMERSET _ NIL
                   WAKEREASON _ T
                   PROCFORM _ FORM
                   RESTARTABLE _ RESTARTFLG
                   PROCPRIORITY _ PRIORITY
                   PROCSTATUS _ \PSTAT.WAITING
                   PROCSYSTEMP _ SYSTEMP
                   PROCAFTEREXIT _ AFTEREXIT
                   PROCBEFOREEXIT _ BEFOREEXIT
                   PROCTTYENTRYFN _ TTYENTRYFN
                   PROCTTYEXITFN _ TTYEXITFN
                   PROCWINDOW _ WINDOW
                   PROCINFOHOOK _ INFOHOOK
                   PROCUSERDATA _ USERPROPS
                   PROCRESTARTFORM _ RESTARTFORM))
          (replace PROCQUEUE of PROC with (\GET.PRIORITY.QUEUE (fetch PROCPRIORITY of PROC)))
          (\SET.PROCESS.NAME PROC (OR NAME (CAR FORM)))
          (UNINTERRUPTABLY
              (SETQ \PROCESSES (CONS PROC \PROCESSES))
              (\INVALIDATE.PROCESS.WINDOW)
              (COND
                 (CREATENOW                                 ; Only create it if we are actually scheduling
                        (\MAKE.PROCESS0 FORM PROC)
                        (OR SUSPENDIT (\RUN.PROCESS PROC)))
                 (T (replace PROCNEVERSTARTED of PROC with T))))
          (COND
             (WINDOW (WINDOWPROP WINDOW 'PROCESS PROC)))
          (RETURN PROC])
```

## (DEL.PROCESS

```
  [LAMBDA (PROC INTERNAL)                                   ; Edited 2-Dec-86 20:35 by bvm:
    (LET ((P (\COERCE.TO.PROCESS PROC)))
         (COND
            (P (if (NEQ P (THIS.PROCESS))
```

```
                    then                                    ; Delete proc in its own context, so that (THIS.PROCESS) is
                                                            ; correct during the unwind
                         (if (NOT (fetch PROCBEINGDELETED of P))
                             then (replace PROCBEINGDELETED of P with T)
                                  (\PROCESS.MAKEFRAME P (FUNCTION \UNWIND.PROCESS)
                                         (LIST P)))
                      else                                  ; delete current process.
                           (replace PROCBEINGDELETED of P with T)
                           (\UNWIND.PROCESS P))
                T])
```

## (PROCESS.RETURN

```
  [LAMBDA (VALUE)                                           (* bvm%: " 4-MAY-83 12:35")
    (RETTO '\MAKE.PROCESS0 VALUE])
```

## (FIND.PROCESS

```
  [LAMBDA (PROC ERRORFLG)                                   ; Edited 12-Oct-87 17:17 by bvm:
```

;; Coerces PROC to a process handle, returning handle if okay; otherwise, if ERRORFLG is set, causes an error, else returns NIL. If ERRORFLG ;; is true, also causes error if proc is not alive

```
    (COND
       [(COND
           ((type? PROCESS PROC)
            (AND (NOT (fetch PROCDELETED of PROC))
                 PROC))
           ((OR (LITATOM PROC)
                (STRINGP PROC))
            (GETHASH PROC \PROCESS.NAME.TABLE]
       (ERRORFLG (ERROR PROC "not a live process"])
```

## (MAP.PROCESSES

```
  [LAMBDA (MAPFN)                                           (* bvm%: "16-JUN-82 16:22")
    (for P in (APPEND \PROCESSES) do (APPLY* MAPFN P (fetch PROCNAME of P)
                                             (fetch PROCFORM of P))
       unless (DEADPROCP P])
```

## (PROCESSP

```
  [LAMBDA (PROC)                                            (* bvm%: " 6-JUL-82 17:30")
    (AND (type? PROCESS PROC)
         (ALIVEPROCP PROC])
```

## (RELPROCESSP

```
  [LAMBDA (PROCHANDLE)                                      (* bvm%: "13-JUN-82 14:39")
    (AND (type? PROCESS PROCHANDLE)
         (DEADPROCP PROCHANDLE])
```

## (RESTART.PROCESS

```
  [LAMBDA (PROC)                                            (* bvm%: "12-Nov-86 17:24")
    (LET ((P (\COERCE.TO.PROCESS PROC)))
         (COND
            (P (UNINTERRUPTABLY
                   (replace WAKEREASON of P with \PROC.RESTARTME)
                   (COND
                      ((EQ P (THIS.PROCESS))
                       (RETTO '\MAKE.PROCESS0 \PROC.RESTARTME))
                      (T (\PROCESS.MAKEFRAME P (FUNCTION RESTART.PROCESS)
                                 (LIST P))
                         P)))])
```

## (WAKE.PROCESS

```
  [LAMBDA (PROC STATUS)                                     (* bvm%: " 4-MAY-83 14:58")
```

;; cause a (possibly) sleeping process to run --- Note that the STATUS will be returned as the value of the BLOCK that put the process to sleep

```
    (DECLARE (GLOBALVARS PSTAT.WAKEUP))
    (PROG ((P (\COERCE.TO.PROCESS PROC)))
          (COND
             (P (UNINTERRUPTABLY
                    [COND
                       ((NEQ (fetch PROCSTATUS of P)
                             \PSTAT.RUNNING)
                        (\RUN.PROCESS P (OR STATUS PSTAT.WAKEUP)))
                       (T (replace WAKEREASON of P with (OR STATUS PSTAT.WAKEUP])
                (RETURN T])
```

## (SUSPEND.PROCESS

```
  [LAMBDA (PROC)                                            (* bvm%: " 4-MAY-83 12:37")
    (PROG [(P (COND
                 (PROC (\COERCE.TO.PROCESS PROC T))
                 (T (THIS.PROCESS]
```

```
          (COND
             ((EQ P (THIS.PROCESS))
              (\PROCESS.GO.TO.SLEEP))
             (T (\SUSPEND.PROCESS P)))
          (RETURN P])
```

## (PROCESS.RESULT

```
  [LAMBDA (PROCESS WAITFORRESULT)                                    (* bvm%: " 1-JUN-83 22:26")
    (SETQ PROCESS (\DTEST PROCESS 'PROCESS))
    (COND
       ((DEADPROCP PROCESS)
        (fetch PROCRESULT of PROCESS))
       (WAITFORRESULT (bind [EVENT _ (OR (fetch PROCFINISHEVENT of PROCESS)
                                         (replace PROCFINISHEVENT of PROCESS
                                            with (CREATE.EVENT (CONCAT (fetch PROCNAME of PROCESS)
                                                                       "#Finish"]
                            until (DEADPROCP PROCESS) do (AWAIT.EVENT EVENT)
                            finally (RETURN (fetch PROCRESULT of PROCESS])
```

## (PROCESS-STATUS

```
  [LAMBDA (POS)                                                      ; Edited 12-Oct-87 17:28 by bvm:
    (LET ((POS (if POS
                   then (if (OR (LITATOM POS)
                                (STRINGP POS))
                            then (FIND.PROCESS POS)
                            else POS)
                   else 2)))
         (AND POS (LET ((STKI (\STACKARGPTR POS)))
                       (do (SELECTQ (fetch (FX FRAMENAME) of STKI)
                                    ((\INTERRUPTFRAME \INTERRUPTED INTERRUPTED \DOINTERRUPTHERE
                                            \PERIODIC.INTERRUPTFRAME ERRORSET)
                                                                     ; Skip over these
                                         (SETQ STKI (fetch (FX CLINK) of STKI)))
                                    ((\GETCHAR \GETKEY \TTYBACKGROUND GETMOUSESTATE MENU.HANDLER)
                                         (RETURN 'IO.WAIT))
                                    ((BLOCK \BACKGROUND AWAIT.EVENT MONITOR.AWAIT.EVENT \PROCESS.GO.TO.SLEEP)
                                                                     ; Forms of blocking
                                         (RETURN 'WAITING))
                                    (RETURN 'RUNNING])
```

## (PROCESS.FINISHEDP

```
  [LAMBDA (PROCESS)                                                  (* bvm%: "17-SEP-82 11:53")
    (SETQ PROCESS (\DTEST PROCESS 'PROCESS))
    (COND
       ((fetch PROCFINISHED of PROCESS))
       ((DEADPROCP PROCESS)
        'ERROR])

)

(DEFINEQ
```

## (THIS.PROCESS

```
  [LAMBDA NIL                                                        (* bvm%: " 4-MAY-83 13:47")
    \RUNNING.PROCESS])
```

## (TTY.PROCESS

```
  [LAMBDA (PROC)                                                     ; Edited  3-May-89 18:21 by atm
    (PROG1 (AND (type? PROCESS \TTY.PROCESS)
                \TTY.PROCESS)
        [COND
           (PROC (PROG ((OLDTTY \TTY.PROCESS)
                        NEWTTY OTHER OLDTTYDS TYPEAHEAD FN)
                       [SETQ NEWTTY (COND
                                       [(EQ PROC T)                  ; Return tty to default tty process
                                        (COND
                                           ((AND OLDTTY (SETQ OTHER (fetch PROCOLDTTYPROC of OLDTTY))
                                                 (OPENWP (WFROMDS (PROCESS.TTY OTHER)
                                                                  T)))
                                                                     ; Only give it to recent tty process if its window is open.  Avoids
                                                                     ; popping up shrunk tedit's, etc.
                                            OTHER)
                                           ((for P in TTY.PROCESS.DEFAULT
                                               when [AND (SETQ OTHER (FIND.PROCESS P))
                                                         (NEQ OTHER OLDTTY)
                                                         (OR (NOT (HASTTYWINDOWP OTHER))
                                                             (OPENWP (WFROMDS (PROCESS.TTY OTHER)
                                                                              T]
                                               do                    ; lets us return it to MOUSE but not to a shrunken EXEC
                                                  (RETURN OTHER)))
                                           ((for P in \PROCESSES
                                               when [AND (NEQ P OLDTTY)
                                                         (OR (NOT (HASTTYWINDOWP P))
```

```
                                                    (OPENWP (WFROMDS (PROCESS.TTY P)
                                                                    T]
                                                 do (RETURN P)))
                                              ((EQ (CAR \PROCESSES)
                                                   OLDTTY)                ; If nothing on TTY.PROCESS.DEFAULT exists, pick something
                                                                          ; random
                                               (CADR \PROCESSES))
                                              (T (CAR \PROCESSES]
                                          ((type? PROCESS PROC)
                                           PROC)
                                          (T (FIND.PROCESS PROC T]
                                (COND
                                   ((fetch PROCDELETED of NEWTTY)         ; Ordinarily would error, but this can easily happen from a
                                                                          ; RESETFORM
                                    (RETURN)))
                                (COND
                                   ((NEQ NEWTTY OLDTTY)
                                    (if (AND OLDTTY (NEQ PROC T))
                                        then                              ; record in new process which process used to be the tty, for use
                                                                          ; of (tty.process t)
                                             (replace PROCOLDTTYPROC of NEWTTY with OLDTTY))
                                    (\CHECKCARET)                         ; gonna switch TTY, take down caret wherever it is
                                    [COND
                                       ((AND (SETQ TYPEAHEAD (bind C while (SETQ C (\GETSYSBUF)) collect C))
                                             OLDTTY)                      ; Save any typeahead that was done while old proc had the tty
                                        (replace PROCTYPEAHEAD of OLDTTY with (NCONC (fetch PROCTYPEAHEAD of OLDTTY)
                                                                                     TYPEAHEAD]
                                    (LET* [(KEYACTION (OR (PROCESSPROP NEWTTY 'KEYACTION)
                                                          \DEFAULTKEYACTION))
                                           (NEWINTERRUPTS (PROCESSPROP NEWTTY 'INTERRUPTS]
                                          (UNINTERRUPTABLY
                                              (COND
                                                 ((AND OLDTTY (SETQ FN (fetch PROCTTYEXITFN of OLDTTY)))
                                                  (CL:FUNCALL FN OLDTTY NEWTTY)))
                                              (SETQ \TTY.PROCESS NEWTTY)
                                              [PROCESSPROP OLDTTY 'INTERRUPTS (LET ((INTERRUPTLIST (fetch (KEYACTION
                                                                                                           INTERRUPTLIST
                                                                                                           )
                                                                                                  of
                                                                                                   \CURRENTKEYACTION
                                                                                                     )))
                                                                                    (if INTERRUPTLIST
                                                                                        then (APPEND INTERRUPTLIST)
                                                                                      else 'OFF]
                                                                                    ; save the old interrupts on the process.
                                              (SETQ \CURRENTKEYACTION KEYACTION)
                                                                                    ; set the new interrupts up.
                                              (AND NEWINTERRUPTS (REPLACE (KEYACTION INTERRUPTLIST) OF \CURRENTKEYACTION
                                                                    WITH (AND (NEQ NEWINTERRUPTS 'OFF)
                                                                              NEWINTERRUPTS)))
                                              (COND
                                                 ((SETQ FN (fetch PROCTTYENTRYFN of NEWTTY))
                                                  (CL:FUNCALL FN NEWTTY OLDTTY)))
                                              (NOTIFY.EVENT \TTY.PROCESS.EVENT))])])
```

## (TTY.PROCESSP

```
[LAMBDA (PROC)                                                            (* bvm%: " 5-MAY-83 18:14")
  (OR (NULL (THIS.PROCESS))
      (EQ (OR PROC (THIS.PROCESS))
          (TTY.PROCESS])
```

## (PROCESS.TTY

```
[LAMBDA (PROC)                                                            (* lmm "20-Jan-86 23:51")
  ;; returns the TTY for a process
  (COND
     ((OR (NULL PROC)
          (EQ (SETQ PROC (\COERCE.TO.PROCESS PROC))
              (THIS.PROCESS)))
      \TERM.OFD)
     (PROC (PROCESS.EVALV PROC '\TERM.OFD])
```

## (GIVE.TTY.PROCESS

```
[LAMBDA (WINDOW)                                                          (* rrb "16-Jul-84 17:53")
  ;; default WINDOWENTRYFN which gives the tty to the process associated with this window and calls its BUTTONEVENTFN
  (OR (WINDOWP WINDOW)
      (\ILLEGAL.ARG WINDOW))
  (PROG ((PROC (WINDOWPROP WINDOW 'PROCESS))
         FN)
        [COND
           (PROC (COND
                    ((DEADPROCP PROC)
                     (WINDOWPROP WINDOW 'PROCESS NIL))
```

```
                    (T (TTY.PROCESS PROC]
          (AND [SETQ FN (COND
                        ((LASTMOUSESTATE (ONLY RIGHT))
                         (fetch RIGHTBUTTONFN of WINDOW))
                        (T (fetch BUTTONEVENTFN of WINDOW]
               (APPLY* FN WINDOW])
```

## (ALLOW.BUTTON.EVENTS

```
[LAMBDA NIL                                                    (* bvm%: "24-JUL-83 15:31")
  (AND (EQ (fetch PROCNAME of (THIS.PROCESS))
           'MOUSE)
       (SPAWN.MOUSE (THIS.PROCESS])
```

## (SPAWN.MOUSE

```
[LAMBDA (INTERNAL)                                             ; Edited  8-May-87 17:34 by bvm
  (UNINTERRUPTABLY
      (PROG ([MOUSEPROC (COND
                          ((AND INTERNAL (EQ (fetch PROCNAME of INTERNAL)
                                             'MOUSE))
                           INTERNAL)
                          (T (FIND.PROCESS 'MOUSE]
            NAME)
            (COND
               (MOUSEPROC (\SET.PROCESS.NAME MOUSEPROC (COND
                                                         ((NOT (FIND.PROCESS 'OLDMOUSE))
                                                             ; First spawned mouse
                                                          'OLDMOUSE)
                                                         ((for I from 2 to (COND
                                                                             (INTERNAL PROCESS.MAXMOUSE)
                                                                             (T MAX.SMALLP))
                                                             unless (FIND.PROCESS (SETQ NAME
                                                                                    (CONCAT 'OLDMOUSE
                                                                                            '%# I)))
                                                             do
                                                             ; Get a unique name
                                                                (RETURN NAME)))
                                                         (T  ; Too many mice, stop before we eat up the whole stack
                                                            (RETURN)))
                                                       T)
                                (replace PROCSYSTEMP of MOUSEPROC with NIL)
                                                             ; Make non systemp in case user wants to kill it

                          ))
               (ADD.PROCESS (LIST '\MOUSE.PROCESS)
                      'NAME
                      'MOUSE
                      'RESTARTABLE
                      'SYSTEM)
               (RETURN T)))])
```

## (\WAIT.FOR.TTY

```
[LAMBDA NIL                                                    (* bvm%: " 5-MAY-83 12:43")
  (until (TTY.PROCESSP) do (AWAIT.EVENT \TTY.PROCESS.EVENT])
```

## (WAIT.FOR.TTY

```
[LAMBDA (MSECS NEEDWINDOW)                                     (* kbr%: "29-Jan-86 12:59")
```

;;; Ensures that current process can take input.  Blocks if necesary until it becomes tty process

```
    (COND
       ((EQ (fetch PROCNAME of (THIS.PROCESS))
            'MOUSE)
        (SPAWN.MOUSE (THIS.PROCESS)))
        ;; Background proc cannot take input, because if we block it, then nobody is listening to the mouse.  So spin off a new background process
        ;; and relegate this one to the tty use                ; Assume mouse-invoked action wants to have the tty
        [OR (TTY.PROCESSP)
            (SETQ \OLDTTY (TTY.PROCESS (THIS.PROCESS]
        T)
       ((TTY.PROCESSP)
        T)
       [\WINDOWWORLD (PROG (WINDOW TIMER)
                           [COND
                              (NEEDWINDOW                      ; Make sure process has a tty window
                                     (OR [OPENWP (SETQ WINDOW (WFROMDS (PROGN (\GETSTREAM T 'INPUT)
                                                                              (TTYDISPLAYSTREAM]
                                         (OPENW WINDOW]
                           [COND
                              (MSECS                           ; Put a time limit on the wait
                                     (SETQ TIMER (SETUPTIMER MSECS]
                           (RETURN (do (AWAIT.EVENT \TTY.PROCESS.EVENT TIMER TIMER)
                                       (COND
                                          ((TTY.PROCESSP)
                                           (RETURN T))
```

```
                                        ((AND TIMER (TIMEREXPIRED? TIMER))
                                         (RETURN NIL]
       (T (TTY.PROCESS (THIS.PROCESS))
          T])

)

(DEFINEQ
```

## (RESET

```
  [LAMBDA NIL                                                    (* bvm%: "10-Nov-86 18:16")
    (PROG ((FX (\MYALINK)))
      LP  [COND
             ((SELECTQ (fetch (FX FRAMENAME) of FX)
                   ((T \MAKE.PROCESS0 \REPEATEDLYEVALQT)
                        T)
                   NIL)
              ;; In process world, try to return to top level exec frame (\REPEATEDLYEVALQT), or to the top of the process, which will decide
              ;; whether to restart or kill the process.  In non-process world, we eventually return to the T frame
              (\SMASHRETURN NIL FX)
              (RETURN \PROC.RESETME))
             ((fetch (FX INVALIDP) of (SETQ FX (fetch (FX CLINK) of FX)))
              (RETURN (printout PROMPTWINDOW .TAB0 0 "Can't find top of stack!!!"]
          (GO LP])
```

## (ERROR!

```
  [LAMBDA NIL                                                    (* bvm%: "12-Nov-86 17:49")
    (if NIL
        then                                                     ; old way--unwind to errorset or top
             [PROG ((FX (\MYALINK))
                    NFX)
               LP  (SELECTQ (fetch (FX FRAMENAME) of FX)
                        (ERRORSET                                ; return from NLSETQ, ERSETQ etc
                                  (\SMASHLINK NIL (fetch (FX CLINK) of FX)
                                         (fetch (FX ALINK) of FX))
                                  (RETURN))
                        (\MAKE.PROCESS0                          ; no ERRORSETs to be found, so return to top-level of process
                             (\SMASHLINK NIL FX FX)
                             (RETURN))
                        (if (fetch (FX INVALIDP) of (SETQ NFX (fetch (FX CLINK) of FX)))
                            then                                 ; return to top.  This can only happen in non-process world
                                 (\SMASHLINK NIL FX FX)
                                 (RETURN)
                          else (SETQ FX NFX)
                               (GO LP]
      else (ABORT)                                               ; If ABORT returns, must have been no CATCH-ABORT, so
                                                                 ; reset to top
           (RETTO '\MAKE.PROCESS0 \PROC.RESETME])

)

(RPAQ? TTY.PROCESS.DEFAULT '(EXEC MOUSE))

(RPAQ? \TTY.PROCESS.EVENT )

(RPAQ? \TTY.PROCESS )

(RPAQ? \PROCESS.NAME.TABLE (HASHARRAY 30 NIL (FUNCTION STRING-EQUAL-HASHBITS)
                                  (FUNCTION STRING-EQUAL)))

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS TTY.PROCESS.DEFAULT \TTY.PROCESS.EVENT \PROCESS.NAME.TABLE)
)

(DEFINEQ
```

## (PROCESSPROP

```
  [LAMBDA ARGS                                                   ; Edited 12-Oct-87 17:40 by bvm:
    (LET ((P (ARG ARGS 1))
          (PROP (ARG ARGS 2))
          NEWVALUE OLDDATA OLDVALUE)                             ; First arg is the process handle or name. It is allowed to be a
                                                                 ; dead process, for benefit of folks retrieving props from a
                                                                 ; process after it dies.
         (AND (OR (type? PROCESS P)
                  (SETQ P (FIND.PROCESS P)))
              (PROG1 (SELECTQ PROP
                          (WINDOW (fetch PROCWINDOW of P))
                          (PRIORITY (fetch PROCPRIORITY of P))
                          (NAME (fetch PROCNAME of P))
                          (RESTARTABLE (fetch RESTARTABLE of P))
                          (FORM (fetch PROCFORM of P))
                          (INFOHOOK (fetch PROCINFOHOOK of P))
                          (AFTEREXIT (fetch PROCAFTEREXIT of P))
                          (BEFOREEXIT (fetch PROCBEFOREEXIT of P))
```

```
                        (TTYENTRYFN (fetch PROCTTYENTRYFN of P))
                        (TTYEXITFN (fetch PROCTTYEXITFN of P))
                        (USERDATA (fetch PROCUSERDATA of P))
                        (RESTARTFORM (fetch PROCRESTARTFORM of P))
                        (SETQ OLDVALUE (LISTGET (SETQ OLDDATA (fetch PROCUSERDATA of P))
                                               PROP)))
                [COND
                   ((> ARGS 2)
                    (SETQ NEWVALUE (ARG ARGS 3))
                    (SELECTQ PROP
                         (WINDOW [replace PROCWINDOW of P with (AND NEWVALUE (SETQ NEWVALUE (\INSUREWINDOW
                                                                                             NEWVALUE]
                                 (if NEWVALUE
                                     then (WINDOWPROP NEWVALUE 'PROCESS P)))
                         (PRIORITY NIL)
                         (NAME (\SET.PROCESS.NAME P NEWVALUE)
                               (\INVALIDATE.PROCESS.WINDOW))
                         (RESTARTABLE (replace RESTARTABLE of P with (SELECTQ NEWVALUE
                                                                              ((NIL NO NEVER)
                                                                                  NIL)
                                                                              ((T YES ALWAYS)
                                                                                  T)
                                                                              (HARDRESET 'HARDRESET)
                                                                              (\ILLEGAL.ARG NEWVALUE))))
                         (FORM)
                         (INFOHOOK (replace PROCINFOHOOK of P with NEWVALUE))
                         (AFTEREXIT (replace PROCAFTEREXIT of P with NEWVALUE))
                         (BEFOREEXIT (replace PROCBEFOREEXIT of P with NEWVALUE))
                         (TTYENTRYFN (replace PROCTTYENTRYFN of P with NEWVALUE))
                         (TTYEXITFN (replace PROCTTYEXITFN of P with NEWVALUE))
                         (USERDATA (replace PROCUSERDATA of P with NEWVALUE))
                         (RESTARTFORM (replace PROCRESTARTFORM of P with NEWVALUE))
                         (COND
                            [(NOT NEWVALUE)                    ; Delete the old value, if any
                             (COND
                                ((EQ (CAR OLDDATA)
                                     PROP)
                                 (replace PROCUSERDATA of P with (CDDR OLDDATA)))
                                (T (for TAIL on (CDR OLDDATA) by (CDDR TAIL) when (EQ (CADR TAIL)
                                                                                      PROP)
                                      do (RPLACD TAIL (CDDDR TAIL))
                                         (RETURN]
                            (OLDDATA (LISTPUT OLDDATA PROP NEWVALUE))
                            (T (replace PROCUSERDATA of P with (LIST PROP NEWVALUE])])
```

## (PROCESS.NAME

```
  [LAMBDA (PROC NAME)                                  ; Edited  8-May-87 17:27 by bvm
    (LET ((P (\COERCE.TO.PROCESS PROC)))
         (AND P (PROG1 (fetch PROCNAME of P)
                       (COND
                          (NAME (\SET.PROCESS.NAME P NAME))))])
```

## (PROCESS.WINDOW

```
  [LAMBDA (PROC WINDOW)                                (* bvm%: "16-JUN-82 16:36")
    ;; Associates WINDOW with PROC, for exec switching
    (LET ((P (\COERCE.TO.PROCESS PROC)))
         (COND
            (P (PROG1 (fetch PROCWINDOW of P)
                      (COND
                         (WINDOW (replace PROCWINDOW of P with (SETQ WINDOW (\INSUREWINDOW WINDOW)))
                                 (WINDOWPROP WINDOW 'PROCESS P)))))])

)

(PUTPROPS PROCESSPROP ARGNAMES (PROC PROP NEWVALUE))

(PUTPROPS ADD.PROCESS ARGNAMES (NIL (FORM . PROPS&VALUES) . U))

;; Temporary

(MOVD? 'PROCESS.RETURN 'KILL.ME NIL T)

(DEFINEQ
```

## (DISMISS

```
  [LAMBDA (MSECSWAIT TIMER NOBLOCK)                    (* bvm%: " 5-Nov-85 10:52")
    (PROG (DTIMER)
          [SETQ DTIMER (COND
                          [MSECSWAIT (SETUPTIMER (IMIN MSECSWAIT MAX.FIXP)
                                                 (OR TIMER (GETRESOURCE \DISMISSTIMER]
                          (TIMER (\DTEST TIMER 'FIXP))
                          (T (RETURN (BLOCK]
          (COND
             ((NOT (THIS.PROCESS))                     ; Process world off
```

```
            (SETQ NOBLOCK T)))
          (do (OR NOBLOCK (\PROCESS.GO.TO.SLEEP NIL DTIMER T)) until (TIMEREXPIRED? DTIMER))
          (OR TIMER (FREERESOURCE \DISMISSTIMER DTIMER)))
    MSECSWAIT])
```

## (BLOCK

```
[LAMBDA (MSECSWAIT TIMER)                                   (* kbr%: " 1-Feb-86 12:12")
```

;; Waits for MSECSWAIT or forever if MSECSWAIT=T. Yields if MSECSWAIT is NIL. TIMER can be given as an alternative for specifying how ;; long to wait.

```
(PROG ((PROC (THIS.PROCESS))
       PQUEUE)
      (RETURN (COND
                 [(type? PROCESS PROC)
                  (COND
                     ((AND (NULL MSECSWAIT)
                           (NULL TIMER))                    ; Only yielding, not going to sleep
                      (UNINTERRUPTABLY
                          (SETQ PQUEUE (fetch PROCQUEUE of PROC))
                          (COND
                             ((NEQ PROC (fetch PQNEXT of PQUEUE))
                              (\MP.ERROR \MP.PROCERROR "Current process is not its queue's NEXT" PROC)))
                          (replace WAKEREASON of PROC with T)
                          (replace PQNEXT of PQUEUE with (fetch NEXTPROCHANDLE of PROC))
                          (replace PQLAST of PQUEUE with PROC)
                          (\RESCHEDULE PROC)))
                     (T (\PROCESS.GO.TO.SLEEP NIL (COND
                                                     (TIMER (\DTEST TIMER 'FIXP))
                                                     ((FIXP MSECSWAIT)
                                                      (IMIN MSECSWAIT MAX.FIXP)))
                              (NEQ TIMER NIL]
                 ((FIXP MSECSWAIT)                          ; Not scheduling;  act like DISMISS
                  (DISMISS MSECSWAIT T)
                  NIL)
                 (T (AND \WINDOWWORLD (WINDOW.MOUSE.HANDLER))
                    (for FN in BACKGROUNDFNS do (SPREADAPPLY* FN))
                    NIL])
```

## (WAITFORINPUT

```
[LAMBDA (N)                                                 (* bvm%: "24-Jul-85 12:21")
  (COND
     [(FIXP N)
      (GLOBALRESOURCE (\DISMISSTIMER)
             (PROG ((NOW (\CLOCK0 \DISMISSTIMER))
                    (N-100 (IDIFFERENCE N 100))
                    ELAPSED)
               LP   (COND
                       ((READP T)
                        (RETURN T))
                       ((NOT (\CLOCKGREATERP NOW N-100))    ; only run background task if at least 100 msecs left
                        (\TTYBACKGROUND))
                       ((\CLOCKGREATERP NOW N)               ; Time's up, return with no input
                        (RETURN)))
                    (GO LP]
     (N                                                     ; Getting OFD avoids time wasted in directory search, leaves
                                                            ; more time for \TTYBACKGROUND
        (bind (STREAM _ (\GETSTREAM N 'INPUT)) until (OR (READP T)
                                                         (READP STREAM))
           do (\TTYBACKGROUND)))
     (T (until (READP T) do (\TTYBACKGROUND])
```

## (\WAITFORSYSBUFP

```
[LAMBDA (N)                                                 (* bvm%: "24-Jul-85 12:22")
  (COND
     [(FIXP N)
      (GLOBALRESOURCE (\DISMISSTIMER)
             (PROG ((NOW (\CLOCK0 \DISMISSTIMER)))
               LP   (COND
                       ((\SYSBUFP)
                        (RETURN T))
                       ((NOT (TTY.PROCESSP))
                        (\WAIT.FOR.TTY))
                       ((\CLOCKGREATERP NOW N)               ; Time's up, return with no input
                        (RETURN))
                       (T (BLOCK)))
                    (GO LP]
     (T (until (\SYSBUFP) do (BLOCK)
                             (\WAIT.FOR.TTY])

)

;; Used to be a GLOBALRESOURCES

(DECLARE%: DONTCOPY
```

```
(DECLARE%: EVAL@COMPILE

[PUTDEF '\DISMISSTIMER 'RESOURCES '(NEW (SETUPTIMER 0]
)
)

(/SETTOPVAL '\\DISMISSTIMER.GLOBALRESOURCE NIL)

(DEFINEQ
```

## (EVAL.AS.PROCESS

```
  [LAMBDA (FORM)                                          (* bvm%: "20-MAY-83 12:00")
    (COND
       ((THIS.PROCESS)
        (ADD.PROCESS FORM 'RESTARTABLE 'NO))
       (T (\EVAL FORM])
```

## (EVAL.IN.TTY.PROCESS

```
  [LAMBDA (FORM WAITFORRESULT)                            (* bvm%: " 5-MAY-83 18:14")
    (COND
       ((TTY.PROCESSP)
        (\EVAL FORM))
       (T (PROCESS.EVAL (TTY.PROCESS)
                 FORM WAITFORRESULT])

)

;; The PROCESS.WAIT macro is an augmentation to BLOCK, waiting for a condition to come true, or a timeout, or a wakeup

(DECLARE%: EVAL@COMPILE

(PUTPROPS PROCESS.WAIT MACRO [(WAITCOND TIMEOUT)
                              (bind ($$TIMEOUT _ (AND TIMEOUT (SETUPTIMER TIMEOUT)))
                                 until (AND $$TIMEOUT (TIMEREXPIRED? $$TIMEOUT))
                                 do (if (SETQ $$VAL WAITCOND)
                                        then (RETURN $$VAL)
                                      else (BLOCK])
)

(DEFINEQ
```

## (PROCESS.READ

```
  [LAMBDA (WINDOW PROMPT CLEAR?)                          (* bvm%: " 5-MAY-83 12:54")
    ;; Special case of PREEMPT.KEYBOARD
    (PROG ((OLDTTY (TTY.PROCESS))
           OLDW)
          (RETURN (PROG1 (NLSETQ (PROGN (TTY.PROCESS (THIS.PROCESS))
                                        [COND
                                           (WINDOW (SETQ OLDW (TTYDISPLAYSTREAM WINDOW))
                                                   (COND
                                                      (CLEAR? (CLEARW WINDOW]
                                        (COND
                                           (PROMPT (PRIN1 PROMPT T)))
                                        (READ T T)))
                         (TTY.PROCESS OLDTTY)
                         (AND OLDW (TTYDISPLAYSTREAM OLDW)))])
```

## (PROCESS.EVALV

```
  [LAMBDA (PROC VAR)                                      (* bvm%: " 8-Jun-85 23:08")
    (LET ((P (\COERCE.TO.PROCESS PROC T))
          ME)
         (COND
            ((OR (NULL (\DTEST VAR 'LITATOM))
                 (EQ VAR T))
             VAR)
            (T [COND
                  ((NEQ P (THIS.PROCESS))
                   (SETQ ME (\MYALINK))
                   (\SMASHLINK NIL (fetch PROCFX of P]
               (PROG1 (\GETBASEPTR (\STKSCAN VAR)
                                   0)
                      (AND ME (\SMASHLINK NIL ME)))])
```

## (PROCESS.EVAL

```
  [LAMBDA (PROC FORM WAITFORRESULT)                       ; Edited  9-Nov-87 18:54 by bvm:
    (DECLARE (LOCALVARS . T))
    (PROG ((P (\COERCE.TO.PROCESS PROC T))
           (ME (THIS.PROCESS)))
          [COND
             ((EQ P ME)
              (RETURN (CL:EVAL FORM]
          (COND
```

```
            (WAITFORRESULT (replace PROCEVAPPLYRESULT of ME with \PSTAT.NORESULT)))
          (\PROCESS.MAKEFRAME P '\PROCESS.EVAL1 (CONS FORM (AND WAITFORRESULT (LIST ME)))
                 T)
          (RETURN (COND
                     (WAITFORRESULT (do (\PROCESS.GO.TO.SLEEP) until (NEQ (fetch PROCEVAPPLYRESULT of ME)
                                                                          \PSTAT.NORESULT))
                            (PROG1 (fetch PROCEVAPPLYRESULT of ME)
                                   (replace PROCEVAPPLYRESULT of ME with \PSTAT.NORESULT])
```

## (\PROCESS.EVAL1

```
[LAMBDA (..PEV-FORM.. ..PEV-PROC..)                          ; Edited 10-Nov-87 14:50 by bvm
```

;; Evaluate the FORM argument and give the result to the calling process. If PROC is nil, then the evaluation is for effect only, and nobody needs ;; to see it.

```
(if ..PEV-PROC..
    then ;; Be careful here that aborting or killing the evaluation still causes the calling process to see a result.
         [LET ((..PEV-RESULT.. :ABORTED))
              (CL:UNWIND-PROTECT
                  (SETQ ..PEV-RESULT.. (\EVAL ..PEV-FORM..))
                  (replace PROCEVAPPLYRESULT of ..PEV-PROC.. with ..PEV-RESULT..)
                  (COND
                     ((NEQ (fetch PROCSTATUS of ..PEV-PROC..)
                           \PSTAT.RUNNING)                   ; Make caller run.
                      (\RUN.PROCESS ..PEV-PROC..)))))]
  else                                                       ; Just eval it.
         (\EVAL ..PEV-FORM..])
```

## (PROCESS.APPLY

```
[LAMBDA (PROC FN ARGS WAITFORRESULT)                         ; Edited  9-Nov-87 18:54 by bvm:
  (DECLARE (LOCALVARS . T))
  (PROG ((P (\COERCE.TO.PROCESS PROC T))
         (ME (THIS.PROCESS)))
        [COND
           ((EQ P ME)
            (RETURN (APPLY FN ARGS]
        (COND
           (WAITFORRESULT (replace PROCEVAPPLYRESULT of ME with \PSTAT.NORESULT)))
        (\PROCESS.MAKEFRAME P '\PROCESS.APPLY1 (LIST* FN ARGS (AND WAITFORRESULT (LIST ME)))
               T)
        (RETURN (COND
                   (WAITFORRESULT (do (\PROCESS.GO.TO.SLEEP) until (NEQ (fetch PROCEVAPPLYRESULT of ME)
                                                                        \PSTAT.NORESULT))
                          (PROG1 (fetch PROCEVAPPLYRESULT of ME)
                                 (replace PROCEVAPPLYRESULT of ME with \PSTAT.NORESULT])
```

## (\PROCESS.APPLY1

```
[LAMBDA (..PEV-FN.. ..PEV-ARGS.. ..PEV-PROC..)               ; Edited  9-Nov-87 18:54 by bvm:
```

;; Apply the FN to the ARGS and give the result to the calling process. If PROC is nil, then the evaluation is for effect only, and nobody needs to ;; see it. Ugly names here are because the Interlisp compiler will make them specvars on account of the UNWIND-PROTECT, and we can't use ;; the XCL compiler in the init.

```
(if ..PEV-PROC..
    then ;; Be careful here that aborting or killing the evaluation still causes the calling process to see a result.
         [LET ((..PEV-RESULT.. :ABORTED))
              (CL:UNWIND-PROTECT
                  (SETQ ..PEV-RESULT.. (APPLY ..PEV-FN.. ..PEV-ARGS..))
                  (replace PROCEVAPPLYRESULT of ..PEV-PROC.. with ..PEV-RESULT..)
                  (COND
                     ((NEQ (fetch PROCSTATUS of ..PEV-PROC..)
                           \PSTAT.RUNNING)                   ; Make caller run.
                      (\RUN.PROCESS ..PEV-PROC..)))))]
  else                                                       ; Just call it.
         (APPLY ..PEV-FN.. ..PEV-ARGS..])

)

;; Standard values for WAKEREASON -- PSTAT.TIMEDOUT is the only public one

(RPAQ PSTAT.WAKEUP "default WakeUp")

(RPAQ PSTAT.TIMEDOUT "{time interval expired}")

(RPAQ PSTAT.QUIT "Quit")

(RPAQ \PSTAT.NORESULT "{no result yet}")

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS PSTAT.WAKEUP PSTAT.TIMEDOUT PSTAT.QUIT \PSTAT.NORESULT)
)
```

```
;; Event stuff

(DECLARE%: DONTCOPY

(DECLARE%: EVAL@COMPILE

(DATATYPE EVENT ((EVENTWAKEUPPENDING FLAG)                    ; True if this event was signaled with nobody waiting on it
                 (NIL BITS 3)
                 (EVENTQUEUETAIL POINTER)                     ; Pointer to last process waiting on this event
                 (EVENTNAME POINTER)                          ; Optional name of EVENT for status window, debugging, etc

                 )
       (ACCESSFNS EVENT ((EVLOCKQUEUETAIL (ffetch EVENTQUEUETAIL of DATUM)
                                 (freplace EVENTQUEUETAIL of DATUM with NEWVALUE)))
                                                              ; Used by both EVENT and MONITORLOCK data

             ))
)

(/DECLAREDATATYPE 'EVENT '(FLAG (BITS 3)
                                POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '4)
)

(/DECLAREDATATYPE 'EVENT '(FLAG (BITS 3)
                                POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '4)

(ADDTOVAR SYSTEMRECLST (DATATYPE EVENT ((EVENTWAKEUPPENDING FLAG)
                                        (NIL BITS 3)
                                        (EVENTQUEUETAIL POINTER)
                                        (EVENTNAME POINTER))))

(DEFINEQ

(CREATE.EVENT
  [LAMBDA (NAME)                                              (* bvm%: " 5-MAY-83 11:00")
    (create EVENT
            EVENTNAME _ NAME])

(NOTIFY.EVENT
  [LAMBDA (EVENT ONCEONLY)                                    (* bvm%: " 3-Jan-85 12:10")
    ;; Wake up any process waiting for EVENT, or only the first one if ONCEONLY is true
    (SETQ EVENT (\DTEST EVENT 'EVENT))
    (PROG (PROC SUCCESS TAIL)
      LP  (UNINTERRUPTABLY
              (COND
                 ((SETQ TAIL (ffetch EVENTQUEUETAIL of EVENT))
                  (SETQ PROC (fetch PROCEVENTLINK of TAIL))
                  [COND
                     ((EQ PROC TAIL)
                      (freplace EVENTQUEUETAIL of EVENT with (SETQ TAIL NIL)))
                     (T (replace PROCEVENTLINK of TAIL with (fetch PROCEVENTLINK of PROC]
                  (replace PROCEVENTLINK of PROC with (replace PROCEVENTORLOCK of PROC with NIL))
                  (\RUN.PROCESS PROC EVENT)
                  (SETQ SUCCESS T))
                 ((NOT SUCCESS)
                  ;; Indicate that a wakeup was signaled, even though nobody was waiting.  Handles most cases where the wakeup would
                  ;; otherwise be lost by occurring between a process's testing a condition and waiting on the event
                  (freplace EVENTWAKEUPPENDING of EVENT with T))))
          (COND
             ((AND TAIL (NOT ONCEONLY))
              (GO LP])

(AWAIT.EVENT
  [LAMBDA (EVENT TIMEOUT TIMERP)                              (* bvm%: " 5-Nov-85 11:09")
    [COND
       (TIMEOUT                                               ; Check args before going uninterruptable
              (SETQ TIMEOUT (COND
                                (TIMERP (\DTEST TIMEOUT 'FIXP))
                                ((TYPENAMEP TIMEOUT 'BIGNUM)
                                 MAX.FIXP)
                                (T (FIX TIMEOUT]
    (\PROCESS.GO.TO.SLEEP (\DTEST EVENT 'EVENT)
          TIMEOUT TIMERP])

(\UNQUEUE.EVENT
  [LAMBDA (PROC EVENT)                                        (* bvm%: " 3-Jan-85 12:34")
```

;; Remove PROC from EVENT's queue. EVENT is an EVENT or MONITORLOCK. Their queues consist of a pointer to the last item in the queue, ;; which in turn points to the first item

```
(PROG ((TAIL (ffetch EVLOCKQUEUETAIL of EVENT))
       NEXT)
      [COND
         ((NOT TAIL)
          (\MP.ERROR \MP.PROCERROR "Process not on its EVENT/MONITOR queue" PROC))
         (T (while (NEQ PROC (SETQ NEXT (ffetch PROCEVENTLINK of TAIL))) do (SETQ TAIL NEXT))
            (COND
               ((EQ PROC TAIL)
                (freplace EVLOCKQUEUETAIL of EVENT with NIL))
               (T (replace PROCEVENTLINK of TAIL with (fetch PROCEVENTLINK of PROC))
                  (COND
                     ((EQ PROC (fetch EVLOCKQUEUETAIL of EVENT))
                      (freplace EVLOCKQUEUETAIL of EVENT with (fetch PROCEVENTLINK of PROC]
      (replace PROCEVENTORLOCK of PROC with NIL)
      (replace PROCEVENTLINK of PROC with NIL])
```

## (**ENQUEUE.EVENT/LOCK**

```
  [LAMBDA (PROC EVLOCK)                                   (* bvm%: " 3-Jan-85 12:15")
```

;;; Enqueue process PROC on EVLOCK's waiting queue. EVLOCK is either an EVENT or a MONITORLOCK

```
    (PROG (TAIL)
          (replace PROCEVENTORLOCK of PROC with EVLOCK)
     ;; Put PROC at end of event or monitorlock's queue. Queue tail is pointed to by a common field in EVENT and MONITORLOCK. The tail itself
     ;; points at the first item in the queue
          (freplace PROCEVENTLINK of PROC with (COND
                                                  ((SETQ TAIL (ffetch EVLOCKQUEUETAIL of EVLOCK))
                                                   (PROG1 (fetch PROCEVENTLINK of TAIL)
                                                          (freplace PROCEVENTLINK of TAIL with PROC)))
                                                  (T PROC)))
          (freplace EVLOCKQUEUETAIL of EVLOCK with PROC])
```

## (**EVENT.DEFPRINT**

```
  [LAMBDA (EVENT STREAM)                                  ; Edited  8-May-87 15:55 by bvm
    (\DEFPRINT.BY.NAME EVENT STREAM (fetch (EVENT EVENTNAME) of EVENT)
           "Event"])

)

(DECLARE%: EVAL@COMPILE

(PUTPROPS AWAIT.CONDITION MACRO [(CONDITION EVNT TIMEOUT TIMERP)
                                 (PROG [($$TIMER TIMEOUT)
                                        ($$EV (\DTEST EVNT 'EVENT]
                                       (DECLARE (LOCALVARS $$TIMER $$EV))
                                   LP  (RETURN (OR CONDITION (COND
                                                               ((NEQ (\PROCESS.GO.TO.SLEEP $$EV $$TIMER
                                                                                           TIMERP)
                                                                     $$EV)
                                                                NIL)
                                                               (T (AND $$TIMER (SETQ $$TIMER T))
                                                                  (GO LP])
)

(RPAQ? \PROCESS.AFTEREXIT.EVENT )

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \PROCESS.AFTEREXIT.EVENT)
)

;; Monitor stuff

(DECLARE%: DONTCOPY

(DECLARE%: EVAL@COMPILE

(DATATYPE MONITORLOCK ((NIL FLAG)
                       (MLOCKPERPROCESS FLAG)                ; Monitor's use by anybody in process lets everyone in that proc
                                                             ; use it, the normal case
                       (NIL BITS 2)
                       (MLOCKQUEUETAIL POINTER)              ; Last process waiting for monitor to become available
                       (MLOCKOWNER POINTER)                  ; Process owning it
                       (MLOCKNAME POINTER)                   ; optional name, for debugging, etc
                       (MLOCKLINK POINTER)                   ; Link to next lock owned by my owner

                       ))
)

(/DECLAREDATATYPE 'MONITORLOCK '(FLAG FLAG (BITS 2)
                                      POINTER POINTER POINTER POINTER)
```

```
        ;; ---field descriptor list elided by lister---
        '8)

(DECLARE%: EVAL@COMPILE

(PUTPROPS .RELEASE.LOCK. MACRO [(LOCK EVENIFNOTMINE)
                                (UNINTERRUPTABLY
                                    [PROG ((OWNER (ffetch MLOCKOWNER of LOCK))
                                           TAIL PREV NEXTPROC)
                                          (COND
                                             ((OR (NULL OWNER)
                                                  (AND (NEQ OWNER (THIS.PROCESS))
                                                       (NOT EVENIFNOTMINE)))
                                              (RETURN)))
                                          (freplace MLOCKOWNER of LOCK with NIL)
                                                                  ; Now remove LOCK from my list of owned locks
                                          [COND
                                             ((EQ (SETQ PREV (fetch PROCOWNEDLOCKS of OWNER))
                                                  LOCK)
                                              (replace PROCOWNEDLOCKS of OWNER with (ffetch MLOCKLINK of LOCK)))
                                             (T (do (COND
                                                       ((NULL PREV)
                                                        (RETURN (\MP.ERROR \MP.PROCERROR "Lock not found among
                                                                                   owner's owned locks" LOCK)))
                                                       [(EQ (fetch MLOCKLINK of PREV)
                                                            LOCK)
                                                        (RETURN (replace MLOCKLINK of PREV
                                                                   with (ffetch MLOCKLINK of LOCK]
                                                       (T (SETQ PREV (fetch MLOCKLINK of PREV]
                                          (freplace MLOCKLINK of LOCK with NIL)
                                          (COND
                                             ((SETQ TAIL (ffetch MLOCKQUEUETAIL of LOCK))
                                              (SETQ NEXTPROC (fetch PROCEVENTLINK of TAIL))
                                              [COND
                                                 ((EQ NEXTPROC TAIL)
                                                                  ; Only one process in queue
                                                  (freplace MLOCKQUEUETAIL of LOCK with NIL))
                                                 (T (replace PROCEVENTLINK of TAIL with (fetch PROCEVENTLINK
                                                                                           of NEXTPROC]
                                              (replace PROCEVENTLINK of NEXTPROC
                                                 with (replace PROCEVENTORLOCK of NEXTPROC with NIL))
                                              (\RUN.PROCESS NEXTPROC LOCK])])

)
)

(/DECLAREDATATYPE 'MONITORLOCK '(FLAG FLAG (BITS 2)
                                  POINTER POINTER POINTER POINTER)
        ;; ---field descriptor list elided by lister---
        '8)

(ADDTOVAR SYSTEMRECLST (DATATYPE MONITORLOCK ((NIL FLAG)
                                              (MLOCKPERPROCESS FLAG)
                                              (NIL BITS 2)
                                              (MLOCKQUEUETAIL POINTER)
                                              (MLOCKOWNER POINTER)
                                              (MLOCKNAME POINTER)
                                              (MLOCKLINK POINTER))))

(DEFINEQ

(OBTAIN.MONITORLOCK
  [LAMBDA (LOCK DONTWAIT UNWINDSAVE)                    ; Edited 24-Feb-87 14:45 by bvm:
    ;; Attempts to acquire lock.  If lock is busy, waits until it is available, unless DONTWAIT is true, in which case it returns NIL immediately.  Returns
    ;; LOCK if it grabbed the lock, T if the current process already had the lock.  If UNWINDSAVE is true, does the appropriate RESETSAVE to release
    ;; the lock on exit of the surrounding RESETLST
    (SETQ LOCK (\DTEST LOCK 'MONITORLOCK))
    (PROG ((PROC (THIS.PROCESS))
           (WASINTERRUPTABLE \INTERRUPTABLE)
           (\INTERRUPTABLE))
      LP   (RETURN (COND
                      ((NULL (fetch MLOCKOWNER of LOCK))            ; Lock is idle
                       [SELECTQ UNWINDSAVE
                            (NIL)
                            (WITH.MONITOR                           ; from WITH.MONITOR macro -- set variable freely to be
                                                                    ; unlocked by SI::MONITOR-UNWIND.  Do it this way rather than
                                                                    ; in caller to reduce the possibility that it won't get unlocked
                                          (SETQ SI::*LOCKED-MONITOR* LOCK))
                            (PROGN                                  ; The normal RESETLST method of ensuring unwind
                                   (RESETSAVE (PROGN LOCK)
                                              '(RELEASE.MONITORLOCK OLDVALUE]
                       (replace MLOCKOWNER of LOCK with PROC)
                       (replace MLOCKLINK of LOCK with (fetch PROCOWNEDLOCKS of PROC))
                                                                    ; Link lock into list of those owned by this process
                       (replace PROCOWNEDLOCKS of PROC with LOCK)   ; return lock for those that care
```

```
                    LOCK)
                   [(EQ (fetch MLOCKOWNER of LOCK)
                        PROC)                                ; My process already owns it
                    (COND
                       ((fetch MLOCKPERPROCESS of LOCK)      ; ok, per-process lock
                        T)
                       (T (ERROR "Trying to acquire lock exclusively owned already by this process" LOCK]
                   ((NOT DONTWAIT)
                    (PROG ((\INTERRUPTABLE WASINTERRUPTABLE))
                          (\PROCESS.GO.TO.SLEEP LOCK))
                    (GO LP])
```

## (CREATE.MONITORLOCK

```
  [LAMBDA (NAME EXCLUSIVE)                                   (* bvm%: "17-MAY-83 17:58")
    (create MONITORLOCK
            MLOCKPERPROCESS _ (NOT EXCLUSIVE)
            MLOCKNAME _ NAME])
```

## (RELEASE.MONITORLOCK

```
  [LAMBDA (LOCK EVENIFNOTMINE)                               (* bvm%: "22-Nov-86 18:40")
    (COND
       ((EQ LOCK 'OLDVALUE)                                  ; Hack for RESETSAVE
        (SETQ LOCK OLDVALUE)))
    (SETQ LOCK (\DTEST LOCK 'MONITORLOCK))
    (.RELEASE.LOCK. LOCK EVENIFNOTMINE])
```

## (SI::MONITOR-UNWIND

```
  [LAMBDA NIL                                                ; Edited 24-Feb-87 14:51 by bvm:
    ;; Cleanup fn for WITH.MONITOR's implicit unwind-protect.
    (if SI::*RESETFORMS*
        then ;; WITH.MONITOR used to be implicit RESETLST, so best to keep it that way--handle RESETSAVEs before releasing lock
             (SI::RESETUNWIND))
    (LET ((LOCK SI::*LOCKED-MONITOR*))
         (if LOCK
             then (SETQ LOCK (\DTEST LOCK 'MONITORLOCK))
                  (.RELEASE.LOCK. LOCK])
```

## (MONITOR.AWAIT.EVENT

```
  [LAMBDA (RELEASELOCK EVENT TIMEOUT TIMERP)                 (* bvm%: " 5-Nov-85 11:10")
    [COND
       (TIMEOUT                                              ; Check args before going uninterruptable
              (SETQ TIMEOUT (COND
                                (TIMERP (\DTEST TIMEOUT 'FIXP))
                                ((TYPENAMEP TIMEOUT 'BIGNUM)
                                 MAX.FIXP)
                                (T (FIX TIMEOUT]
    (RELEASE.MONITORLOCK RELEASELOCK)
    (PROG1 (\PROCESS.GO.TO.SLEEP (\DTEST EVENT 'EVENT)
                  TIMEOUT TIMERP)
           (OBTAIN.MONITORLOCK RELEASELOCK])
```

## (\MONITORLOCK.DEFPRINT

```
  [LAMBDA (LOCK STREAM)                                      ; Edited  8-May-87 15:55 by bvm
    (\DEFPRINT.BY.NAME LOCK STREAM (fetch (MONITORLOCK MLOCKNAME) of LOCK)
           "Lock"])

)

(DECLARE%: EVAL@COMPILE

(PUTPROPS WITH.MONITOR MACRO [(LOCK . FORMS)
                              (LET (SI::*LOCKED-MONITOR* SI::*RESETFORMS*)
                                   (DECLARE (CL:SPECIAL SI::*LOCKED-MONITOR* SI::*RESETFORMS*))
                                   (CL:UNWIND-PROTECT
                                       (PROGN (OBTAIN.MONITORLOCK LOCK NIL 'WITH.MONITOR) . FORMS)
                                       (SI::MONITOR-UNWIND))])

(PUTPROPS WITH.FAST.MONITOR MACRO [(LOCK . FORMS)
                                   (UNINTERRUPTABLY
                                       ([LAMBDA (UNLOCK)
                                          (PROG1 (PROGN . FORMS)
                                                 (AND (NEQ UNLOCK T)
                                                      (RELEASE.MONITORLOCK UNLOCK)))]
                                        (OBTAIN.MONITORLOCK LOCK)))])
)

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: DOEVAL@COMPILE DONTCOPY
```

```
(SPECVARS \BACKGROUND)
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \IGNORE.BACKGROUND)
)

;; END EXPORTED DEFINITIONS

(RPAQ? \BACKGROUND NIL)

(RPAQ? \IGNORE.BACKGROUND T)

(DEFINEQ

(\MAKE.PROCESS0
  [LAMBDA (FORM HANDLE)                                    ; Edited 18-Jan-87 17:25 by bvm:
    (DECLARE (LOCALVARS . T)
           (SPECVARS %#FORM# HELPFLAG \CURRENTDISPLAYLINE \#DISPLAYLINES \LINEBUF.OFD *STANDARD-INPUT*
                  *READTABLE* \PRIMREADTABLE \PRIMTERMTABLE \PRIMTERMSA TtyDisplayStream \TERM.OFD \TTYWINDOW
                  *STANDARD-OUTPUT* \INTERRUPTABLE READBUF *CURRENT-PROCESS* SI::*RESETFORMS* *DRIBBLE-OUTPUT*)
           (GLOBALVARS \DEFAULTLINEBUF \DEFAULTTTYDISPLAYSTREAM))
    (PROG ((%#FORM# FORM)
           (*CURRENT-PROCESS* HANDLE)
           (SI::*RESETFORMS*)
           (HELPFLAG (AND HELPFLAG 'BREAK!))
           (\CURRENTDISPLAYLINE 0)
           (\#DISPLAYLINES 40)
           (\LINEBUF.OFD (OR \DEFAULTLINEBUF \LINEBUF.OFD))
           (*READTABLE* *READTABLE*)
           (\PRIMTERMTABLE \PRIMTERMTABLE)
           (\PRIMTERMSA \PRIMTERMSA)
           (TtyDisplayStream \DEFAULTTTYDISPLAYSTREAM)
           (\INTERRUPTABLE)
           (\TTYWINDOW)
           (READBUF)
           \TERM.OFD *STANDARD-OUTPUT* *STANDARD-INPUT* RESULT TMP)
                                                           ; HELPFLAG set to ensure breaks occur.  Proc can rebind if
                                                           ; desired
      ;; \TTYWINDOW is currently just a place to hold onto the WINDOW of the TtyDisplayStream in case user closes same and then someone prints
      ;; to TtyDisplayStream
          (\MISCAPPLY* (FUNCTION \PROCESS.MOVEFRAME))      ; Move me to the boonies
          [SETQ *STANDARD-OUTPUT* (SETQ \TERM.OFD (COND
                                                    (TtyDisplayStream (\GETOFD TtyDisplayStream 'OUTPUT))
                                                    (T          ; For init time, before LLDISPLAY sets up
                                                       (GETTOPVAL '\TERM.OFD]
          (SETQ *STANDARD-INPUT* \LINEBUF.OFD)
          (PROGN ;; Make this proc use a piece of its PROCESS handle as the binding place for *DRIBBLE-OUTPUT*.  This lets its survive a
                 ;; HARDRESET.  The extra third arg to \SETFVARSLOT is so that the compiler will create a fvar slot for *DRIBBLE-OUTPUT* in
                 ;; the first place.
                 (\SETFVARSLOT '*DRIBBLE-OUTPUT* (LOCF (fetch PROCDRIBBLEOUTPUT of HANDLE))
                        *DRIBBLE-OUTPUT*))
          (\MAKE.PROCESS1 HANDLE)
          (SETQ \INTERRUPTABLE T)                          ; Safe to go interruptable now
          [if (SETQ TMP (fetch PROCHARDRESETINFO of HANDLE))
              then                                         ; new style cleanup from HARDRESET
                   (if (EQ TMP 'ERROR)
                       then (printout T T "***Warning: errors occurred recovering stack for process "
                                  (fetch PROCNAME of HANDLE)
                                  "; cleanups not run" T)
                     else (SELECTQ (\HARDRESET-CLEANUP HANDLE)
                                (ERROR (printout T T "***Warning: errors occurred running cleanups for process "
                                             (fetch PROCNAME of HANDLE)
                                             T))
                                (T                         ; ok
                                   )
                                (GO ABORT]
          (if (fetch PROCFINISHED of HANDLE)
              then                                         ; Happens after a HARDRESET -- proc was restarted only long
                                                           ; enough to clean up after itself
                   (GO DIE))
       LP                                                  ; Unwind anything left from last invocation
          (if SI::*RESETFORMS*
              then (LET [(RESETSTATE (COND
                                        ((EQ RESULT \PROC.RESETME)
                                                           ; From RESET
                                         'RESET)
                                        (T 'HARDRESET]
                        (DECLARE (SPECVARS RESETSTATE))
                        (SI::RESETUNWIND)))
          (\PROCESS.RELEASE.LOCKS HANDLE)
          (SETQ RESULT (\EVAL %#FORM#))
          (if (EQ RESULT \PROC.KILLME)
              then                                         ; from \UNWIND.PROCESS
```

```
                (GO DIE)
           elseif (EQ RESULT \PROC.RESTARTME)
             then                                                    ; from RESTART.PROCESS
                (SETQ %#FORM# (OR (fetch PROCRESTARTFORM of HANDLE)
                                  %#FORM#))
                (GO LP)
           elseif (EQ RESULT \PROC.RESETME)
             then                                                    ; RESET or ERROR! -- maybe restart
                (if (EQ (fetch RESTARTABLE of HANDLE)
                        T)
                    then                                           ; Autorestart on errors
                         (SETQ %#FORM# (OR (fetch PROCRESTARTFORM of HANDLE)
                                           %#FORM#))
                         (GO LP))
           else                                                    ; normal termination
                (replace PROCRESULT of HANDLE with RESULT)
                (replace PROCFINISHED of HANDLE with 'NORMAL)
                (GO DIE))
      ABORT
           (printout PROMPTWINDOW T (fetch PROCNAME of HANDLE)
                  " aborted.")
           (replace PROCFINISHED of HANDLE with 'ERROR)
      DIE (if SI::*RESETFORMS*
              then                                                 ; Every process has implicit RESETLST at top, so do it
                   (LET ((RESETSTATE 'RESET))
                        (DECLARE (SPECVARS RESETSTATE))
                        (SI::RESETUNWIND)))
           (DRIBBLE)                                               ; Close dribble file, if any
           (LET ((EVENT (fetch PROCFINISHEVENT of HANDLE)))
                (AND EVENT (NOTIFY.EVENT EVENT)))
           (COND
              ((EQ HANDLE (TTY.PROCESS))                           ; It is possible that while unwinding, so this check happens very
                                                                   ; late
               (TTY.PROCESS T)))
           (\PROCESS.RELEASE.LOCKS HANDLE)
           (\PROCESS.GO.TO.SLEEP NIL NIL NIL T])
```

## (\MAKE.PROCESS1

```
  [LAMBDA (PROC)                                                   (* bvm%: " 8-Jun-85 23:14")
```

;;; Called by \MAKE.PROCESS0 to set up PROC's initial handle and then return to its caller, usually ADD.PROCESS --- we have here a partial
;;; exchange of stack pointers: PROC gets pointer to \MAKE.PROCESS0 frame, \MAKE.PROCESS0 points to T, we return to former parent of
;;; \MAKE.PROCESS0; the only use count that changes is the T frame, which now has one more user

```
    (UNINTERRUPTABLY
        (LET ((MP0 (\MYALINK))
              (TOP (\STACKARGPTR T))
              MP0CALLER)
             [COND
                ((NEQ 0 (fetch PROCFX of PROC))                    ; Should never happen, but let's be consistent with stackp use
                 (\DECUSECOUNT (fetch PROCFX of PROC]
             (SETQ MP0CALLER (fetch (FX ALINK) of MP0))
             (replace PROCFX of PROC with MP0)                     ; Fix proc handle to return to \MAKE.PROCESS0
             (replace (FX ACLINK) of MP0 with TOP)                 ; Detach \MAKE.PROCESS0 from the ADD.PROCESS stack
             (\INCUSECOUNT TOP)
             (\RESUME MP0CALLER)                                   ; Make me return to the caller of \MAKE.PROCESS0
             NIL))])
```

## (\PROCESS.MOVEFRAME

```
  [LAMBDA NIL                                                      (* bvm%: " 8-Jun-85 22:30")
```

;; Called in misc context to move a frame to a big free area

```
    (FLIPCURSORBAR 12)
    (PROG ((OLDFRAME (fetch MiscFXP of \InterfacePage))
           NXT NEW FRAMESIZE BFSIZE RESIDUAL FREESIZE FXSIZE BLINK INITSIZE)
          (SETQ BLINK (fetch (FX DUMMYBF) of OLDFRAME))
          [SETQ FRAMESIZE (IPLUS (SETQ FXSIZE (fetch (FX SIZE) of OLDFRAME))
                                 (SETQ BFSIZE (COND
                                                 ((OR (fetch (BF RESIDUAL) of BLINK)
                                                      (SETQ RESIDUAL (NEQ (fetch (BF USECNT) of BLINK)
                                                                          0)))
                                                  WORDSPERCELL)
                                                 (T (fetch (BF SIZE) of BLINK]
          (SETQ NEW (\FREESTACKBLOCK (SETQ FREESIZE (IPLUS FRAMESIZE PROC.FREESPACESIZE))
                                     OLDFRAME))                    ; Find a free stack block
          [COND
             ((type? FSB (SETQ NXT (IPLUS NEW FREESIZE)))
              ;; \FREESTACKBLOCK normally sticks a free block after the block it returns.  We will massage them together
              (add FREESIZE (fetch (FSB SIZE) of NXT]
          (SETQ INITSIZE (FLOOR (LRSH (IDIFFERENCE FREESIZE FRAMESIZE)
                                      1)
                                WORDSPERCELL))                     ; Size of free block to go before
          (COND
             ((EVENP (IPLUS NEW INITSIZE BFSIZE)
```

```
                    WORDSPERQUAD)                                   ; FX must be odd-quad aligned
            (add INITSIZE WORDSPERCELL)))
         (\MAKEFREEBLOCK NEW INITSIZE)
         (add NEW INITSIZE)
         (SETQ FREESIZE (IDIFFERENCE FREESIZE INITSIZE))
         (\BLT (ADDSTACKBASE NEW)
               (ADDSTACKBASE (IDIFFERENCE OLDFRAME BFSIZE))
               FRAMESIZE)                                           ; Copy FX and BF into middle of new free area
         (COND
            (RESIDUAL (replace (BF RESIDUAL) of NEW with T))
            ((NOT (fetch (BF RESIDUAL) of BLINK))                   ; Point new BF at itself
             (replace (BF IVAR) of (IPLUS NEW (IDIFFERENCE BFSIZE WORDSPERCELL)) with NEW)))
         (add NEW BFSIZE)                                           ; now NEW points to the FX
         (replace (FX NEXTBLOCK) of NEW with (SETQ NXT (IPLUS NEW FXSIZE)))
         [replace (FX BLINK) of NEW with (COND
                                           (RESIDUAL                ; Point at real bf
                                                  (fetch (FX BLINK) of OLDFRAME))
                                           (T (IDIFFERENCE NEW WORDSPERCELL]
         [COND
            ((AND (fetch (FX VALIDNAMETABLE) of NEW)
                  (EQ (fetch (FX NAMETABHI) of NEW)
                      \STACKHI))
             (CHECK ([LAMBDA (N)
                       (AND (IGREATERP N OLDFRAME)
                            (ILESSP N (fetch (FX NEXTBLOCK) of OLDFRAME]
                     (fetch (FX NAMETABLO) of OLDFRAME)))
             (add (fetch (FX NAMETABLO) of NEW)
                  (IDIFFERENCE NEW OLDFRAME]
         (\MAKEFREEBLOCK NXT (IDIFFERENCE FREESIZE FRAMESIZE))      ; Install free block after frame
         (COND
            (RESIDUAL (\MAKEFREEBLOCK OLDFRAME (IDIFFERENCE FRAMESIZE WORDSPERCELL)))
            (T (\MAKEFREEBLOCK (IDIFFERENCE OLDFRAME BFSIZE)
                       FRAMESIZE)))                                 ; Finally free up the original frame
      OUT (replace MiscFXP of \InterfacePage with NEW)
         (FLIPCURSORBAR 12)                                         ; Restore cursor
         (RETURN NEW])
```

## (**RELEASE.PROCESS**

```
[LAMBDA (PROC KILLIT RESTARTFLG)                                    ; Edited 1-Jun-88 15:38 by bvm
```

;; Disentangle PROC from process land. If KILLIT is true, free all resources associated with it, since we are about to delete it. RESTARTFLG is ;; when killing a process at HARDRESET. Must be called uninterruptably.

```
(PROG ((EVENT (fetch PROCEVENTORLOCK of PROC))
       FX WINDOW)
      (CHECK (NULL \INTERRUPTABLE))
      (COND
         ((NEQ (SETQ FX (fetch PROCFX of PROC))
               0)
          (\DECUSECOUNT FX)
          (replace PROCFX of PROC with 0)))
      (COND
         (EVENT (\UNQUEUE.EVENT PROC EVENT)))
      (COND
         ((fetch PROCTIMERSET of PROC)
          (\UNQUEUE.TIMER PROC T)))
      (COND
         [KILLIT (for OTHER in \PROCESSES when (EQ (fetch PROCOLDTTYPROC of OTHER)
                                                   PROC)
                    do                                              ; remove links to the dead proc from others
                       (replace PROCOLDTTYPROC of OTHER with NIL))
                 (replace PROCOLDTTYPROC of PROC with NIL)
                 [COND
                    ((NOT RESTARTFLG)                               ; From PROCESSWORLD on HARDRESET. In this case,
                                                                    ; \processes is being iterated over, so we don't want to suffer the
                                                                    ; DREMOVE bug.
                     (SETQ \PROCESSES (DREMOVE PROC \PROCESSES]
                 (REMHASH (fetch PROCNAME of PROC)
                          \PROCESS.NAME.TABLE)
                 (\INVALIDATE.PROCESS.WINDOW)
                 (replace PROCDELETED of PROC with T)
                 (replace PROCSTATUS of PROC with \PSTAT.DELETED)
                 (replace PROCFORM of PROC with (replace PROCRESTARTFORM of PROC
                                                         with (replace PROCQUEUE of PROC with NIL)))
                 (COND
                    ((SETQ WINDOW (fetch PROCWINDOW of PROC))       ; Break link to proc's window
                     (replace PROCWINDOW of PROC with NIL)
                     (WINDOWPROP WINDOW 'PROCESS NIL]
            (T (replace PROCSTATUS of PROC with \PSTAT.WAITING)
               (replace PROCTIMERSET of PROC with NIL)))
      (replace NEXTPROCHANDLE of PROC with NIL])
```

## (**UNWIND.PROCESS**

```
[LAMBDA (P)                                                         ; Edited 2-Dec-86 20:35 by bvm:
   (OR (fetch PROCFINISHED of P)
       (replace PROCFINISHED of P with 'DELETED))
```

```
    (replace PROCBEINGDELETED of P with T)
    (RETTO '\MAKE.PROCESS0 \PROC.KILLME])
```

## (\MAYBEBLOCK

```
  [LAMBDA NIL                                                    (* bvm%: "21-JUN-83 16:01")
    (COND
       (\INTERRUPTABLE (BLOCK])
```

## (\BACKGROUND.PROCESS

```
  [LAMBDA NIL                                                    ; Edited 28-Jul-2023 21:01 by lmm
                                                                 (* bvm%: "24-JUL-83 15:35")
    (PROG ((\BACKGROUND \IGNORE.BACKGROUND))
          (DECLARE (SPECVARS \BACKGROUND)
                   (GLOBALVARS \IGNORE.BACKGROUND))
      LP  (SETQ \BACKGROUND \IGNORE.BACKGROUND)
          (for FN in BACKGROUNDFNS do (SPREADAPPLY* FN))
          (BLOCK)
          (GO LP])
```

## (\MOUSE.PROCESS

```
  [LAMBDA NIL                                                    ; Edited 10-Nov-87 11:18 by bvm:
    (DECLARE (SPECVARS \OLDTTY \MOUSEBUSY))
    (PROG (\OLDTTY \MOUSEBUSY OTHERMOUSE)
      LP  [COND
             ((NEQ (fetch PROCNAME of (THIS.PROCESS))
                   'MOUSE)                                       ; A new mouse process sprung up while we were hung
              (COND
                 ((AND (SETQ OTHERMOUSE (FIND.PROCESS 'MOUSE))
                       (PROCESS.EVALV OTHERMOUSE '\MOUSEBUSY))   ; The other mouse is still busy, so we can't kill it.  Die instead
                  (PROCESS.RETURN))
                 (T (COND
                       (OTHERMOUSE                               ; Kill off the mouse process that took our place
                              (\SET.PROCESS.NAME OTHERMOUSE "DeadMouse" T)
                                                                 ; Have to change its name, since we are about to become the
                                                                 ; unique MOUSE proc, and the DEL.PROCESS does not take
                                                                 ; effect immediately.
                              (DEL.PROCESS OTHERMOUSE)
                              (SETQ OTHERMOUSE)                  ; Don't inadvertantly hold a pointer to this dead process

                              ))
                    (replace PROCSYSTEMP of (THIS.PROCESS) with T)
                    (\SET.PROCESS.NAME (THIS.PROCESS)
                           'MOUSE T]
          (COND
             (\WINDOWWORLD (WINDOW.MOUSE.HANDLER)))
          (COND
             ((TTY.PROCESSP)                                     ; Give up the tty if we still have it
              (TTY.PROCESS (COND
                              ((NEQ \OLDTTY (THIS.PROCESS))
                               \OLDTTY)
                              (T T)))
              (SETQ \OLDTTY)))
          (replace PROCTYPEAHEAD of (THIS.PROCESS) with NIL)     ; No sense keeping around this typeahead
          (BLOCK)
          (GO LP])
```

## (\TIMER.PROCESS

```
  [LAMBDA NIL                                                    (* bvm%: " 1-AUG-83 15:17")
    ;; This process runs at default priority and tests for processes that have timed out
    (PROG ((\INTERRUPTABLE NIL)
           (HEAD \TIMERQHEAD)
           PROC)
      LP   (COND
             ((AND (SETQ PROC (fetch PROCTIMERLINK of HEAD))
                   (TIMEREXPIRED? (fetch PROCWAKEUPTIMER of PROC)))
              (\RUN.PROCESS PROC PSTAT.TIMEDOUT))
             (T (BLOCK)))
          (GO LP])
```

## (\PROCESS.RELEASE.LOCKS

```
  [LAMBDA (P)                                                    (* bvm%: "12-Nov-86 18:07")
    (while (fetch PROCOWNEDLOCKS of P) do (RELEASE.MONITORLOCK (fetch PROCOWNEDLOCKS of P])
```

## (\SET.PROCESS.NAME

```
  [LAMBDA (PROC NEWNAME INTERNAL)                                ; Edited 28-Jan-93 17:44 by jds
```

;;; Changes proc's name to be newname. Unless INTERNAL is true, the name is checked for validity and is coerced to one not in use by any active ;;; process

```
    [COND
```

```
       ((NOT INTERNAL)                                          ; check name
        (PROG NIL
          RETRY
               (SELECTQ (TYPENAME NEWNAME)
                    ((LITATOM NEW-ATOM STRINGP))
                    (LISTP (SETQ NEWNAME (CAR NEWNAME))
                           (GO RETRY))
                    (SETQ NEWNAME (MKSTRING NEWNAME)))
               (COND
                  ((OR (NULL NEWNAME)
                       (EQ NEWNAME T))
                   (SETQ NEWNAME (ERROR "Illegal Process Name" NEWNAME))
                   (GO RETRY]
    (UNINTERRUPTABLY
        [COND
           ((AND (NOT INTERNAL)
                 (FIND.PROCESS NEWNAME))                        ; Proc by this name exists, so make another name
            (for I from 2 bind (FIRSTNAME _ NEWNAME) while (FIND.PROCESS (SETQ NEWNAME (CONCAT FIRSTNAME "#" I]
        (LET ((OLDNAME (FETCH PROCNAME OF PROC)))
             (IF OLDNAME
                 THEN (REMHASH OLDNAME \PROCESS.NAME.TABLE))
             (PUTHASH NEWNAME PROC \PROCESS.NAME.TABLE)
             (replace PROCNAME of PROC with NEWNAME)
             NEWNAME))])
```

## (\PROCESS.DEFPRINT

```
  [LAMBDA (PROC STREAM)                                         ; Edited  8-May-87 15:54 by bvm
    ;; Print process using its name, for example, #<Process MOUSE/76,5432>
    (\DEFPRINT.BY.NAME PROC STREAM (fetch PROCNAME of PROC)
           "Process"])

)

(DEFINEQ
```

## (\START.PROCESSES

```
  [LAMBDA NIL                                                   (* bvm%: " 2-MAY-83 12:30")
    (UNINTERRUPTABLY
        (\RESCHEDULE %#SCHEDULER#))])
```

## (\PROCESS.GO.TO.SLEEP

```
  [LAMBDA (EVLOCK TIMEOUT TIMERP DELETEFLG)                     (* bvm%: " 3-Jan-85 12:34")
    ;; puts the current process to sleep.  EVLOCK is a lock or event to wait on, or NIL for neither.  TIMEOUT is optional timeout to wake up if we
    ;; haven't been woken any other way;  monitor locks do not get timeouts.  TIMERP=T means TIMEOUT is an absolute timer rather than an interval.
    ;; TIMEOUT=T means continue using the timer from the last time we went to sleep.  DELETEFLG means never to return.
    (UNINTERRUPTABLY
        [PROG ((PROC (THIS.PROCESS))
               HEAD TAIL PREV)
              (OR PROC (RETURN (BLOCK)))
              (COND
                 ((AND (type? EVENT EVLOCK)
                       (fetch EVENTWAKEUPPENDING of EVLOCK))    ; Missed a wakeup for this event, take it now
                  (replace EVENTWAKEUPPENDING of EVLOCK with NIL)
                  (RETURN EVLOCK)))
              (replace PROCSTATUS of PROC with \PSTAT.WAITING)
              (SETQ HEAD (fetch PROCQUEUE of PROC))             ; Now remove PROC from its run queue
              (SETQ PREV (fetch PQLAST of HEAD))
              [COND
                 [(EQ PROC PREV)                                ; Nobody left at this level
                  (COND
                     ((EQ PROC (fetch PQNEXT of HEAD))
                      (replace PQLAST of HEAD with (replace PQNEXT of HEAD with NIL)))
                     (T (\MP.ERROR \MP.PROCERROR "Inconsistent process queue state"]
                 (T (replace NEXTPROCHANDLE of PREV with (replace PQNEXT of HEAD with (OR (fetch NEXTPROCHANDLE
                                                                                              of PROC)
                                                                                          (\MP.ERROR \MP.PROCERROR
                                                                                                 "Running process
                                                                                                 has no NEXT
                                                                                                 pointer" PROC]
              (replace NEXTPROCHANDLE of PROC with NIL)
              (COND
                 (EVLOCK (\ENQUEUE.EVENT/LOCK PROC EVLOCK)))
              (replace PROCTIMERSET of PROC with (COND
                                                    (TIMEOUT [COND
                                                                ((NEQ TIMEOUT T)
                                                                 (replace PROCWAKEUPTIMER of PROC
                                                                    with (COND
                                                                            (TIMERP TIMEOUT)
                                                                            (T (SETUPTIMER TIMEOUT
                                                                                           (fetch PROCTIMERBOX
                                                                                              of PROC]
                                                                (\ENQUEUE.TIMER PROC)
                                                                T)))
```

```
            (RETURN (\RESCHEDULE (COND
                                    (DELETEFLG (\RELEASE.PROCESS PROC T)
                                           NIL)
                                    (T PROC])])
```

## (\PROC.RESUME

```
[LAMBDA (FRAME OLDFX)                                          (* bvm%: " 6-Oct-86 14:22")
```

;; Diddles caller so that it returns to FRAME. If OLDFX is non-NIL, it is released. Do it in this order so that the current stack is always valid

```
(replace (FX ACLINK) of (\MYALINK) with FRAME)
(AND OLDFX (\DECUSECOUNT OLDFX])
```

## (\RUN.PROCESS

```
[LAMBDA (PROC REASON BRUTALLY)                                 ; Edited 6-Apr-92 11:39 by jds
```

;; Cause PROC to be placed in the runnable state, with REASON as the value to return from the call to a waiting function

```
(PROG ((PQUEUE (fetch PROCQUEUE of PROC))
       (EVENT (fetch PROCEVENTORLOCK of PROC))
       PREV NEXT)
      (COND
         ((AND (EQ (fetch PROCSTATUS of PROC)
                   \PSTAT.RUNNING)
               (NOT BRUTALLY))
          (ERROR "Attempt to run already running process" PROC)))
      (COND
         ((fetch (PROCESS PROCDELETED) of PROC)
```

;; Process has ended; don't bother restarting it.

;; This used to test PROCFINISHED, but that caused dying processes to hang around holding monitorlocks (JDS, fixing AR 11505)

```
          NIL)
         ((EQ (fetch PROCSTATUS of PROC)
              \PSTAT.DELETED)                                  ; Process has been deleted somehow; don't bother restarting it.
          NIL)
         (T                                                    ; Go ahead and restart the process.
            (UNINTERRUPTABLY
               (COND
                  (EVENT (\UNQUEUE.EVENT PROC EVENT)))
               (COND
                  ((fetch PROCTIMERSET of PROC)
                   (\UNQUEUE.TIMER PROC)))
               (SETQ PREV (fetch PQLAST of PQUEUE))
               (COND
                  [(NOT PREV)                                  ; PROC will be the only process at this level
                   (replace PQNEXT of PQUEUE with (replace PQLAST of PQUEUE
                                                     with (replace NEXTPROCHANDLE of PROC with PROC]
                  [\PROC.RUN.NEXT.FLG (SETQ NEXT (fetch PQNEXT of PQUEUE))
                         (replace NEXTPROCHANDLE of PROC with (fetch NEXTPROCHANDLE of NEXT))
                         (replace NEXTPROCHANDLE of NEXT with PROC)
                         (COND
                            ((EQ NEXT PREV)
                             (replace PQLAST of PQUEUE with PROC]
                  (T (replace NEXTPROCHANDLE of PROC with (fetch NEXTPROCHANDLE of PREV))
                     (replace NEXTPROCHANDLE of PREV with PROC)
                     (replace PQLAST of PQUEUE with PROC)))
               (replace PROCSTATUS of PROC with \PSTAT.RUNNING)
               (replace WAKEREASON of PROC with REASON))])
```

## (\SUSPEND.PROCESS

```
[LAMBDA (PROC EVENT)                                           (* bvm%: " 3-Jan-85 12:35")
```

;;; Suspends PROC, not the running process, waiting on EVENT, or forever if EVENT = NIL

```
(UNINTERRUPTABLY
    [PROG (PQHEAD PREV OLDEVENT NEXT LAST)
          [COND
             ((EQ (fetch PROCSTATUS of PROC)
                  \PSTAT.RUNNING)
```

;; PROC is now running, so put it to sleep with no reason to wake. This is a simplification of \PROCESS.GO.TO.SLEEP

```
              (replace PROCSTATUS of PROC with \PSTAT.WAITING)
              (SETQ PQHEAD (fetch PROCQUEUE of PROC))          ; Now remove PROC from its run queue
              (SETQ PREV (SETQ LAST (fetch PQLAST of PQHEAD)))
              [do (SETQ NEXT (fetch NEXTPROCHANDLE of PREV))
                  (COND
                     ((EQ NEXT PROC)
                      [COND
                         [(NEQ NEXT PREV)
                          (replace NEXTPROCHANDLE of PREV with (fetch NEXTPROCHANDLE of PROC))
                          (COND
                             ((EQ PROC (fetch PQLAST of PQHEAD))
                              (replace PQLAST of PQHEAD with PREV]
                         (T                                    ; Nobody left at this level
                            (replace PQLAST of PQHEAD with (replace PQNEXT of PQHEAD with NIL]
                      (RETURN)))
```

```
                    (COND
                       ((EQ (SETQ PREV NEXT)
                            LAST)
                        (\MP.ERROR \MP.PROCERROR "Can't find running process in its queue"]
                  (replace NEXTPROCHANDLE of PROC with NIL))
                 (T                                                ; Not running, so just keep it from waking up
                    (COND
                       ((fetch PROCTIMERSET of PROC)
                        (\UNQUEUE.TIMER PROC)))
                    (COND
                       ((SETQ OLDEVENT (fetch PROCEVENTORLOCK of PROC))
                        (COND
                           ((NEQ OLDEVENT EVENT)
                            (\UNQUEUE.EVENT PROC OLDEVENT))
                           (T                                      ; Already queued for proper event
                              (SETQ EVENT]
            (COND
               (EVENT (\ENQUEUE.EVENT/LOCK PROC EVENT])])
```

## (\UNQUEUE.TIMER

```
[LAMBDA (PROC NOERROR)                                             (* bvm%: "31-JUL-83 16:29")
```

;; Remove PROC from the timer queue

```
(PROG ((PREV \TIMERQHEAD))
  LP  (COND
         ((EQ (fetch PROCTIMERLINK of PREV)
              PROC)
          (replace PROCTIMERLINK of PREV with (fetch PROCTIMERLINK of PROC)))
         ((SETQ PREV (fetch PROCTIMERLINK of PREV))
          (GO LP))
         ((NULL NOERROR)
          (ERROR "Process not found on timer queue" PROC)))
      (replace PROCTIMERLINK of PROC with NIL)
      (replace PROCTIMERSET of PROC with NIL])
```

## (\ENQUEUE.TIMER

```
[LAMBDA (PROC)                                                     (* bvm%: " 7-SEP-83 13:48")
```

;; Place PROC on the timer queue. Queue is ordered by timeout, so that the first item will timeout first

```
(UNINTERRUPTABLY
    (PROG ((PREV \TIMERQHEAD)
           (NEXT (fetch PROCTIMERLINK of \TIMERQHEAD)))
          [COND
             (NEXT (bind (TIMER _ \PROCTIMER.SCRATCH) first (\BOXIPLUS (\BOXIDIFFERENCE TIMER TIMER)
                                                                       (fetch PROCWAKEUPTIMER of PROC))
                         while (AND NEXT (IGREATERP (\BOXIDIFFERENCE TIMER (fetch PROCWAKEUPTIMER of NEXT))
                                                    0))
                         do                                        ; NEXT will timeout before PROC, so keep going.
                            (\BOXIPLUS TIMER (fetch PROCWAKEUPTIMER of NEXT))
                                                                   ; Restore TIMER
                            (SETQ NEXT (fetch PROCTIMERLINK of (SETQ PREV NEXT]
```

;; PROC goes between PREV and NEXT

```
          (replace PROCTIMERLINK of PROC with NEXT)
          (replace PROCTIMERLINK of PREV with PROC)))])
```

## (\GET.PRIORITY.QUEUE

```
[LAMBDA (PRIORITY)                                                 (* bvm%: "29-APR-83 18:37")
  (PROG ((HEAD \HIGHEST.PRIORITY.QUEUE)
         PREV PQ)
        [COND
           ((NULL HEAD)
            (RETURN (SETQ \HIGHEST.PRIORITY.QUEUE (create PROCESSQUEUE
                                                          PQPRIORITY _ PRIORITY]
    LP  (COND
           ((EQ (fetch PQPRIORITY of HEAD)
                PRIORITY)
            (RETURN HEAD))
           ((IGREATERP (fetch PQPRIORITY of HEAD)
                       PRIORITY)
            (SETQ HEAD (fetch PQLOWER of (SETQ PREV HEAD)))
            (GO LP)))
        (SETQ PQ (create PROCESSQUEUE
                         PQPRIORITY _ PRIORITY
                         PQHIGHER _ PREV
                         PQLOWER _ HEAD))
        (COND
           (PREV (replace PQLOWER of PREV with PQ))
           (T (SETQ \HIGHEST.PRIORITY.QUEUE PQ)))
        (RETURN PQ])

)

(DECLARE%: DONTCOPY
```

```
(DECLARE%: EVAL@COMPILE

(PUTPROPS \RESCHEDULE MACRO [LAMBDA (OLDPROC)
                                    ;; Causes process switch, saving current context in OLDPROC's handle, or nowhere if OLDPROC is NIL.
                                    ;; Must be called uninterruptably!
                                    (PROG (PQUEUE PROC)
                                      TOP
                                     ;; Maybe check for events here?
                                          (SETQ PQUEUE \HIGHEST.PRIORITY.QUEUE)
                                      LP  (COND
                                             ((SETQ PROC (fetch PQNEXT of PQUEUE))
                                              [COND
                                                 ((NEQ PROC OLDPROC) ; Yes, there is a process switch required here.  Below is roughly
                                                                     ; the body of RESUME
                                                  (LET ((TOFX (fetch PROCFX of PROC))
                                                        FROMFX)
                                                       (COND
                                                          ((fetch (FX INVALIDP) of TOFX)
                                                           (\MP.ERROR \MP.STACKRELEASED "Process's stack has been
                                                                  released!" PROC)))
                                                       (SETQ \RUNNING.PROCESS PROC)
                                                       (replace PROCFX of PROC with 0)
                                                       (\PROC.RESUME TOFX (COND
                                                                             (OLDPROC
                                                                              (SETQ FROMFX (fetch PROCFX
                                                                                             of OLDPROC))
                                                                              (COND
                                                                                 ((NOT (fetch (FX INVALIDP)
                                                                                          of FROMFX))
                                                                                  ; Release stack pointer of OLDPROC if it hasn't been yet.
                                                                                  ; should never happen
                                                                                   (\DECUSECOUNT FROMFX)))
                                                                              (replace PROCFX of OLDPROC
                                                                                 with (\MYALINK))
                                                                              NIL)
                                                                             (T
                                                                              ; no OLDPROC to resume later, so jettison caller
                                                                                (\MYALINK]
                                                   (RETURN (fetch WAKEREASON of PROC)))
                                                  ((SETQ PQUEUE (fetch PQLOWER of PQUEUE))
                                                   (GO LP))
                                                  (T                         ; nobody runnable, wait for events
                                                     (\MP.ERROR \MP.PROCERROR "No runnable process!!" OLDPROC)
                                                     (GO TOP])
)
)

(DEFINEQ

(\PROCESS.INIT
  [LAMBDA (DONTRESET)                                          (* lmm "13-Sep-84 15:03")
    (COND
       ((CCODEP '\PROC.CODEFORTFRAME)
        (\DEFINEDEVICE NIL (create FDEV
                                   DEVICENAME _ 'PROCESS
                                   EVENTFN _ (FUNCTION \PROCESS.EVENTFN)
                                   DIRECTORYNAMEP _ 'NILL
                                   HOSTNAMEP _ 'NILL))
        (\LOCKFN '\PROC.CODEFORTFRAME)
        (/PUTD '\CODEFORTFRAME (GETD '\PROC.CODEFORTFRAME)
               T)
        (MOVD 'BLOCK '\BACKGROUND)
        (OR DONTRESET (HARDRESET])


(\PROCESS.EVENTFN
  [LAMBDA (DEV EVENTNAME)                                      (* bvm%: " 3-Apr-84 12:01")
    (SELECTQ EVENTNAME
         ((AFTERLOGOUT AFTERSYSOUT AFTERMAKESYS AFTERSAVEVM)
              [for PROC in (APPEND \PROCESSES) when (AND (ALIVEPROCP PROC)
                                                         (NEQ PROC (THIS.PROCESS)))
                 bind ACTION do                                ; What does this process want done for it after exit?
                              (SELECTQ (SETQ ACTION (fetch PROCAFTEREXIT of PROC))
                                   (DELETE (DEL.PROCESS PROC))
                                   (SUSPEND (SUSPEND.PROCESS PROC))
                                   (COND
                                      ((type? EVENT ACTION)      ; Cause PROC to wait on this event
                                       (\SUSPEND.PROCESS PROC ACTION))
                                      ((NEQ (fetch PROCNAME of PROC)
                                            '\TIMER.PROCESS)
                                       ;; Suspend process until system after exit events have run.  This also has the side effect of eventually
                                       ;; waking any process waiting on a timer, important since the timer is garbage over exit
                                        (\SUSPEND.PROCESS PROC \PROCESS.AFTEREXIT.EVENT])
```

```
        ((BEFOREMAKESYS BEFORELOGOUT BEFORESYSOUT))
        NIL])
```

## (\PROCESS.BEFORE.LOGOUT

```
(LAMBDA NIL                                                    ; Edited 24-Feb-87 14:53 by bvm:
```

;;; Make sure we don't log out until processes that asked to run to completion actually finish

```
    (RESETLST
        [PROG (W)
          RETRY
              (for PROC in \PROCESSES do (COND
                                             ((EQ (fetch PROCBEFOREEXIT of PROC)
                                                  'DON'T)
                                              (ALLOW.BUTTON.EVENTS)
                                                               ; in case logout called from mouse process, want to make sure
                                                               ; user can get a PSW!
                                              [COND
                                                 ((NOT W)
                                                  (RESETSAVE NIL (LIST 'CLOSEW
                                                                       (SETQ W (CREATEW '(260 247 453 173)
                                                                                        "Waiting for process(es) to
                                                                                        finish"]
                                              (printout W T "Waiting for process " (fetch PROCNAME of PROC)
                                                     " to finish..." T " [Use the process status window to kill it
                                                       if you really don't want to wait]" T)
                                              (PROCESS.RESULT PROC T)
                                                               ; Wait for it to finish
                                              (GO RETRY])])
```

## (\PROCESS.AFTER.EXIT

```
(LAMBDA (FLG)                                                  (* bvm%: " 4-Jan-85 12:49")
```

;;; Stuff to do after the system after exit eventfns are finished but before we release to the user

```
    (NOTIFY.EVENT \PROCESS.AFTEREXIT.EVENT)
    (SETQ \PROC.READY T])
```

## (\PROCESS.RESET.TIMERS

```
(LAMBDA NIL                                                    ; Edited  8-May-87 17:42 by bvm
```

```
    ;; Called when the time is up in the air -- clears timers on \SYSTEMTIMERVARS and wakes any process waiting only on a timer
    (for TIMER in \SYSTEMTIMERVARS bind UNITS do [SETQ UNITS (COND
                                                                ((LISTP TIMER)
                                                                 (PROG1 (CADR TIMER)
                                                                     (SETQ TIMER (CAR TIMER)))]
                                                    (SETUPTIMER 0 (COND
                                                                    ((LITATOM TIMER)
                                                                     (GETTOPVAL TIMER))
                                                                    (T TIMER))
                                                                UNITS))
    (for PROC in \PROCESSES when (AND (EQ (fetch PROCSTATUS of PROC)
                                          \PSTAT.WAITING)
                                      (fetch PROCTIMERSET of PROC)
                                      (NOT (fetch PROCEVENTORLOCK of PROC)))
       do (\RUN.PROCESS PROC])
```

## (\PROC.AFTER.WINDOWWORLD

```
(LAMBDA NIL                                                    (* kbr%: " 1-Feb-86 12:12")
  (PROG [(EXECPROC (FIND.PROCESS 'EXEC]
        (COND
           ((AND EXECPROC (TYPENAMEP \TopLevelTtyWindow 'WINDOW))
            (replace PROCWINDOW of EXECPROC with \TopLevelTtyWindow)
            (WINDOWPROP \TopLevelTtyWindow 'PROCESS EXECPROC)))
        (COND
           ([AND \WINDOWWORLD (NOT (FIND.PROCESS 'MOUSE]
            (ADD.PROCESS '(\MOUSE.PROCESS)
                   'NAME
                   'MOUSE
                   'RESTARTABLE
                   'SYSTEM
                   'SCHEDULE T])
```

## (\TURN.ON.PROCESSES

```
(LAMBDA NIL                                                    (* bvm%: "12-Nov-86 18:18")
  (for P in \PROCESSES do ;; CLEARSTK after HARDRESET did not get the process handles, so smash them now
                          (replace PROCFX of P with 0))
  (COND
     ((OR AUTOPROCESSFLG (EQ (ASKUSER NIL NIL "^D -- run process scheduler? " NIL)
                             'Y))
      [COND
```

```
           ((AND NIL (LISTP RESETVARSLST))                    ; Better unwind these now, since this RESETVARSLST binding
                                                              ; will become invisible
            (RESETRESTORE NIL 'RESET]
         (PROCESSWORLD T)))
    'OK])

)

;; Redefinitions

(DEFINEQ
```

## (\PROC.CODEFORTFRAME

```
  [LAMBDA NIL                                                 ; Edited 10-Jan-91 16:42 by jds
    (\CALLME 'T)
    (SETQ \RUNNING.PROCESS)
    (CLEARSTK '**CLEAR**)
    (\KEYBOARDON)                                             ; restore keyboard after hard reset.  Ideally, this should be done
                                                              ; earlier.
    [COND
       ((NEQ (\TURN.ON.PROCESSES)
             'OK)
        (while T do (\MP.ERROR \MP.TOPUNWOUND "Unexpected (RETTO T)"]
    ;; Normally never get here.  There's a hack in \TURN.ON.PROCESSES that lets you run without processes, but I'm not sure you can even do that
    ;; any more.  The OK test is to catch inadvertant (RETTO T) calls
    (INITIALEVALQT)
    (PROG NIL
      LP  (\REPEATEDLYEVALQT)
          (GO LP])
```

## (\PROC.REPEATEDLYEVALQT

```
  [LAMBDA NIL                                                 (* bvm%: "20-Jun-84 17:15")
    (DECLARE (GLOBALVARS \TopLevelTtyWindow))
    (\CALLME '\REPEATEDLYEVALQT)
    (INITIALEVALQT)
    (PROG NIL
          (TTYDISPLAYSTREAM \TopLevelTtyWindow)
          (OUTPUT T)
          (INPUT T)
      LP  (\RESETSYSTEMSTATE)
          (EVALQT)
          (GO LP])

)

;; switching stacks

(DEFINEQ
```

## (BREAK.PROCESS

```
  [LAMBDA (PROC)                                              (* bvm%: "25-JUL-83 17:36")
    (PROG ((P (\COERCE.TO.PROCESS PROC)))
          (COND
             ((EQ P (THIS.PROCESS))
              (\DOHELPINTERRUPT1))
             (T (\PROCESS.MAKEFRAME P (FUNCTION \DOHELPINTERRUPT1])
```

## (\SELECTPROCESS

```
  [LAMBDA (TITLE)                                             (* bvm%: "30-Sep-86 21:22")
    (LET ((TTYNAME (fetch PROCNAME of (TTY.PROCESS)))
          (ME (fetch PROCNAME of (THIS.PROCESS)))
          PROCNAMES NAME)
         ;; Construct list of all processes.  Put the running process and the tty process at the top for ease of recognition
         (SETQ PROCNAMES (CONS TTYNAME (for PROC in \PROCESSES unless [OR (EQ (SETQ NAME (fetch PROCNAME
                                                                                    of PROC))
                                                                              TTYNAME)
                                                                          (EQ NAME ME)
                                                                          (AND (fetch PROCSYSTEMP of PROC)
                                                                               (NEQ NAME 'MOUSE]
                                                  collect NAME)))
         (if (NEQ ME TTYNAME)
             then (push PROCNAMES ME))
         (NCONC PROCNAMES (for PROC in \PROCESSES collect (fetch PROCNAME of PROC)
                             unless (FMEMB (fetch PROCNAME of PROC)
                                           PROCNAMES)))
         [PROGN                                               ; Tag the running and tty procs
                (RPLACA PROCNAMES (LIST (CONCAT ME " *run")
                                        (LIST 'QUOTE ME)))
                (if (NEQ ME TTYNAME)
                    then (RPLACA (CDR PROCNAMES)
                                 (LIST (CONCAT TTYNAME " *tty")
                                       (LIST 'QUOTE TTYNAME]
```

```
         (LET ((MOUSEITEM "[Spawn Mouse]"))
              (if [NOT (SETQ NAME (MENU (create MENU
                                                ITEMS _ (CONS MOUSEITEM PROCNAMES)
                                                TITLE _ TITLE
                                                CENTERFLG _ T
                                                MENUFONT _ INTERRUPTMENUFONT]
                    then NIL
                  elseif (EQ NAME MOUSEITEM)
                    then (SPAWN.MOUSE)
                         NIL
                  else (FIND.PROCESS NAME])
```

## (\PROCESS.MAKEFRAME

```
  [LAMBDA (PROC FN ARGS FLG)                                    (* bvm%: " 5-Feb-85 13:09")
```

;;; Builds a frame to call FN with ARGS on top of PROC. Returns NIL if it can't right now. FN must have no pvars or fvars

```
    (UNINTERRUPTABLY
        (PROG ((FRAME (fetch PROCFX of PROC))
               (FN&ARGS (CONS FN ARGS))
               NEWFRAME)
              [COND
                 ((ILESSP FRAME (fetch (IFPAGE StackBase) of \InterfacePage))
                                                                ; This is the test used in \CAUSEINTERRUPT, but actually, we
                                                                ; could afford to test \INTERRUPTABLE here
                  (RETURN (COND
                             ((EQ FRAME 0)
                              (\MP.ERROR \MP.PROCERROR "PROC confused: trying to call a fn in a nonexistent
                                     process" FN))
                             (T (\MP.ERROR \MP.PROCERROR "PROC confused: a process other than the running one is
                                      in uninterruptable region" FRAME]
              [COND
                 ((SETQ NEWFRAME (\MISCAPPLY* (FUNCTION \PROCESS.MAKEFRAME0)
                                              FRAME FN&ARGS))
                  ;; Note that FN&ARGS was consed up before entering \MISCAPPLY* in case the CONS causes a NEWPAGE, which uses the
                  ;; misc context also

                  )
                 (T                                             ; Should never happen -- error occurs inside
                                                                ; \PROCESS.MAKEFRAME0 first
                    (RETURN (COND
                               (FLG (\MP.ERROR \MP.PROCERROR "Can't build frame for process call" FN]
              (COND
                 ((NEQ (fetch PROCSTATUS of PROC)
                       \PSTAT.RUNNING)
                  (\RUN.PROCESS PROC)))
              (replace PROCFX of PROC with NEWFRAME)
              (RETURN T)))])
```

## (\PROCESS.MAKEFRAME0

```
  [LAMBDA (FRAME FN&ARGS)                                       (* bvm%: " 6-Oct-86 14:22")
    (PROG ((ARGS (CDR FN&ARGS))
           (FN (CAR FN&ARGS))
           FREE NXT NXTEND)
          (SETQ NXT (fetch (FX NEXTBLOCK) of FRAME))
          (CHECK (fetch (FX CHECKED) of FRAME)
                 (type? FSB NXT))
          (SETQ NXTEND (IPLUS NXT (fetch (FSB SIZE) of NXT)))
          [while (type? FSB NXTEND) do (SETQ NXTEND (IPLUS NXTEND (fetch (FSB SIZE) of NXTEND]
          (RETURN (OR (\MAKEFRAME FN NXT NXTEND FRAME FRAME ARGS)
                      (\MAKEFRAME FN (SETQ FREE (\FREESTACKBLOCK (IPLUS (PROG1 (fetch (FNHEADER STKMIN)
                                                                                  of (fetch (LITATOM DEFPOINTER)
                                                                                       of FN))
                                                                                ; Stack needed to call this fn

                                                                                )
                                                                              (PROG1 (UNFOLD 20 WORDSPERCELL)
                                                                                ; Extra slop

                                                                                ))
                                                                        FRAME))
                                  (IPLUS FREE (fetch (FSB SIZE) of FREE))
                                  FRAME FRAME ARGS)
                      (\MP.ERROR \MP.PROCNOFRAME "Failed to build frame for PROCESS use" FN])

)

(RPAQ? %#MYHANDLE# )

(RPAQ? %#SCHEDULER# )

(RPAQ? \RUNNING.PROCESS )

(RPAQ? \PROCESSES )
```

```
(RPAQ? PROCESS.MAXMOUSE 5)

(RPAQ? PROC.FREESPACESIZE 1024)

(RPAQ? AUTOPROCESSFLG T)

(RPAQ? BACKGROUNDFNS )

(RPAQ? \TIMERQHEAD )

(RPAQ? \HIGHEST.PRIORITY.QUEUE )

(RPAQ? PROC.DEFAULT.PRIORITY 2)

(RPAQ? \DEFAULTLINEBUF )

(RPAQ? \DEFAULTTTYDISPLAYSTREAM )

(RPAQ? \PROCTIMER.SCRATCH (NCREATE 'FIXP))

(RPAQ? TOPW )

(RPAQ? \PROC.RUN.NEXT.FLG )

(RPAQ? \PROC.READY T)

(ADDTOVAR \SYSTEMCACHEVARS \PROC.READY)

(ADDTOVAR \SYSTEMTIMERVARS (\LASTUSERACTION SECONDS))

(RPAQ \PROC.RESTARTME "{restart flag}")

(RPAQ \PROC.RESETME "{reset flag}")

(RPAQ \PROC.KILLME "{abort flag}")

(DECLARE%: DONTCOPY
```

;; FOLLOWING DEFINITIONS EXPORTED

```
(DECLARE%: EVAL@COMPILE

(PUTPROPS THIS.PROCESS MACRO (NIL \RUNNING.PROCESS))

(PUTPROPS TTY.PROCESS MACRO [X (COND
                                  ((CAR X)
                                   'IGNOREMACRO)
                                  (T '\TTY.PROCESS])

(PUTPROPS TTY.PROCESSP MACRO [X (COND
                                    ((CAR X)
                                     'IGNOREMACRO)
                                    (T '(OR (NULL (THIS.PROCESS))
                                            (EQ (THIS.PROCESS)
                                                (TTY.PROCESS])
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \RUNNING.PROCESS \TTY.PROCESS \PROC.RESTARTME \PROC.RESETME \PROC.ABORTME)
)
```

;; END EXPORTED DEFINITIONS

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \PROCESSES PROC.FREESPACESIZE %#SCHEDULER# PROCESS.MAXMOUSE AUTOPROCESSFLG BACKGROUNDFNS
       \TopLevelTtyWindow \PROC.READY)
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \TIMERQHEAD \PROCTIMER.SCRATCH \HIGHEST.PRIORITY.QUEUE PROC.DEFAULT.PRIORITY \PROC.RUN.NEXT.FLG
       \SYSTEMTIMERVARS)
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS ALIVEPROCP MACRO ((p)
                            (NOT (DEADPROCP p))))

(PUTPROPS DEADPROCP MACRO ((p)
                           (fetch PROCDELETED of p)))

(PUTPROPS \COERCE.TO.PROCESS MACRO [OPENLAMBDA (P ERRORFLG)
                                    (COND
```

```
                                        ((AND (type? PROCESS P)
                                              (NOT (fetch PROCDELETED of P)))
                                         P)
                                        (T (FIND.PROCESS P ERRORFLG])
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)
```

;; Debugging

```
(DEFINEQ
```

## (\CHECK.PQUEUE

```
  [LAMBDA (P THISP)                                          (* bvm%: "21-Jun-84 11:41")
    [COND
       ((type? PROCESS P)
        (SETQ P (fetch PROCQUEUE of P]
    (OR (PROG ((PREV (fetch PQLAST of P))
               (NEXT (fetch PQNEXT of P))
               X)
              [COND
                 ((NULL PREV)
                  (RETURN (COND
                              ((NULL NEXT)
                               T)
                              (T (printout T P " has a LAST = " PREV " but no NEXT" T)
                                 NIL]
              (COND
                 ((NEQ (fetch NEXTPROCHANDLE of PREV)
                       NEXT)
                  (printout T "Last=" PREV " points at " (fetch NEXTPROCHANDLE of PREV)
                         " but NEXT=" NEXT T)
                  (RETURN)))
              (COND
                 ((AND THISP (NEQ NEXT (THIS.PROCESS)))
                  (printout T "NEXT=" NEXT " but running process = " (THIS.PROCESS)
                          T)
                  (RETURN)))
              (SETQ X (fetch NEXTPROCHANDLE of NEXT))
              (SETQ PREV NEXT)
        LP    (COND
                 ((NULL X)
                  (printout T "Successor of " PREV " is NIL" T)
                  (RETURN)))
              (COND
                 ((EQ X NEXT)                                ; The end
                  (COND
                     ((NEQ PREV (fetch PQLAST of P))
                      (printout T "Predecessor of NEXT = " NEXT " is " PREV " which is not LAST" T)
                      (RETURN)))
                  (RETURN T)))
              (SETQ X (fetch NEXTPROCHANDLE of (SETQ PREV X)))
              (GO LP))
        (RESETVARS ((\RUNNING.PROCESS))                      ; Inhibit process switch
                   (RETURN (HELP])

)

(DEFINEQ
```

## (PPROC

```
  [LAMBDA (PROC FILE)                                        (* bvm%: "10-MAY-83 22:59")
                                                             ; show a process, or many
    (COND
       (PROC (PPROC1 PROC FILE))
       (T (PROG ((NOW (CLOCK 0))
                 (PQ \HIGHEST.PRIORITY.QUEUE)
                 DONE P1)
                (printout FILE "   name" .FR 21 "prty" "  state  (run reason)" T)
           LP   [COND
                   ((SETQ P1 (fetch PQNEXT of PQ))
                    (for (P _ P1) do (PPROC1 P FILE NOW)
                                     (push DONE P)
                       repeatuntil (EQ (SETQ P (fetch NEXTPROCHANDLE of P))
                                       P1]
                (COND
                   ((SETQ PQ (fetch PQLOWER of PQ))
                    (GO LP)))
                (printout FILE "  - - -" T 22 "TimeLeft  WakeCondition" T)
                (for (P _ \TIMERQHEAD) while (SETQ P (fetch PROCTIMERLINK of P)) do (PPROC1 P FILE NOW)
                                                                                    (push DONE P))
                (for P in \PROCESSES unless (FMEMB P DONE) do (PPROC1 P FILE NOW])
```

## (PPROCWINDOW

```
[LAMBDA (W)                                                    (* bvm%: " 6-MAY-83 13:05")
  (OR W (SETQ W (CREATEW NIL "Detailed process status")))
  (WINDOWPROP W 'BUTTONEVENTFN (FUNCTION PPROCREPAINTFN))
  (WINDOWPROP W 'REPAINTFN (FUNCTION PPROCREPAINTFN))
  (WINDOWPROP W 'SCROLLFN (FUNCTION SCROLLBYREPAINTFN))
  (WINDOWPROP W 'RESHAPEFN (FUNCTION PPROCRESHAPEFN))
  (WINDOWPROP W 'PPROCHEIGHT (WINDOWPROP W 'HEIGHT))
  (DSPRIGHTMARGIN 32000 W)
  W])
```

## (PPROCREPAINTFN

```
[LAMBDA (WINDOW REGION)                                        (* bvm%: " 4-MAY-83 12:06")
  [COND
     (REGION                                                   ; As repaintfn
            (MOVETO 0 (WINDOWPROP WINDOW 'PPROCSTART)
                   WINDOW)
            (DSPFILL NIL 0 NIL WINDOW)
            (PPROC NIL WINDOW))
     (T                                                        ; As buttoneventfn
        (COND
           ((LASTMOUSESTATE (NOT UP))
            (CLEARW WINDOW)
            (WINDOWPROP WINDOW 'PPROCSTART (DSPYPOSITION NIL WINDOW))
            (PPROC NIL WINDOW]
  (WINDOWPROP WINDOW 'EXTENT (PPROCEXTENT WINDOW])
```

## (PPROCRESHAPEFN

```
[LAMBDA (WINDOW OLDCONTENTS REGION)                            (* bvm%: "22-JUN-83 10:24")
  (WINDOWPROP WINDOW 'PPROCHEIGHT (WINDOWPROP WINDOW 'HEIGHT))
  (DSPRIGHTMARGIN 32000 WINDOW)
  (RESHAPEBYREPAINTFN WINDOW OLDCONTENTS REGION])
```

## (PPROCEXTENT

```
[LAMBDA (WINDOW)                                               (* bvm%: "10-MAY-83 22:59")
  (PROG [(H (ITIMES (IPLUS 3 (LENGTH \PROCESSES))
                   (IMINUS (DSPLINEFEED NIL WINDOW]
        (RETURN (create REGION
                        LEFT _ 0
                        BOTTOM _ (IDIFFERENCE (WINDOWPROP WINDOW 'PPROCHEIGHT)
                                              H)
                        WIDTH _ -1
                        HEIGHT _ H])
```

## (PPROC1

```
[LAMBDA (PROC FILE NOW)                                        (* bvm%: "10-MAY-83 22:58")
  (PROG (EVLOCK TIMELEFT NAME)
        (PRIN1 (COND
                  ((DEADPROCP PROC)
                   '*)
                  ((EQ PROC (TTY.PROCESS))
                   '%#)
                  (T " "))
               FILE)
        (PRIN1 (COND
                  ((fetch PROCSYSTEMP of PROC)
                   '+)
                  (T " "))
               FILE)
        (printout FILE (fetch PROCNAME of PROC)
               20
               (fetch PROCPRIORITY of PROC)
               %,)
        [COND
           ((EQ PROC (THIS.PROCESS))
            (printout FILE "running "))
           ((EQ (fetch PROCSTATUS of PROC)
                \PSTAT.RUNNING)
            (printout FILE "runnable (" (fetch WAKEREASON of PROC)
                   ")"))
           (T (COND
                 ((NOT (fetch PROCTIMERSET of PROC))
                  (PRIN1 "(forever)" FILE))
                 ((IGEQ [SETQ TIMELEFT (IDIFFERENCE (fetch PROCWAKEUPTIMER of PROC)
                                                    (OR NOW (SETQ NOW (CLOCK 0]
                        0)
                  (printout FILE .I8 TIMELEFT))
                 (T (PRIN1 "(expired)" FILE)))
              (TAB 32 T FILE)
              (COND
                 ((SETQ EVLOCK (fetch PROCEVENTORLOCK of PROC))
                  (printout FILE (COND
```

```
                                    ((type? MONITORLOCK EVLOCK)
                                     (SETQ NAME (fetch MLOCKNAME of EVLOCK))
                                     "lock ")
                                    (T (SETQ NAME (fetch EVENTNAME of EVLOCK))
                                       "event "))
                              (OR NAME "unnamed")))
                     (T (printout FILE "blocked"]
          (TERPRI FILE])
```

## (PROCESS.STATUS.WINDOW

```
  [LAMBDA (WHERE)                                          ; Edited 12-Oct-87 18:13 by bvm:
    (PROG ((PROCS (for P in \PROCESSES collect (fetch PROCNAME of P)))
           PMENU HEIGHT WIDTH LEFT BOTTOM REG)
          (SETQ PMENU (create MENU
                              ITEMS _ PROCS
                              WHENSELECTEDFN _ (FUNCTION \PSW.SELECTED)
                              MENUFONT _ (FONTCREATE 'GACHA 10)
                              CENTERFLG _ T))
          (OR PROCOPMENU (SETQ PROCOPMENU
                               (create MENU
                                       ITEMS _ '(BT WHO? KILL BTV KBD_ RESTART BTV* INFO WAKE BTV! BREAK SUSPEND)
                                       WHENSELECTEDFN _ (FUNCTION \PSWOP.SELECTED)
                                       CENTERFLG _ T
                                       MENUCOLUMNS _ 3)))
          (SETQ HEIGHT (HEIGHTIFWINDOW (+ (fetch IMAGEHEIGHT of PMENU)
                                          (fetch IMAGEHEIGHT of PROCOPMENU)
                                          4)))
          [SETQ WIDTH (WIDTHIFWINDOW (MAX (fetch IMAGEWIDTH of PMENU)
                                          (fetch IMAGEWIDTH of PROCOPMENU]
          [COND
             [(AND (WINDOWP PROCESS.STATUS.WINDOW)
                   (EQ WHERE T))
              (SETQ REG (WINDOWPROP PROCESS.STATUS.WINDOW 'REGION))
              (SETQ LEFT (fetch LEFT of REG))
              (COND
                 ((> (+ (SETQ BOTTOM (fetch BOTTOM of REG))
                        HEIGHT)
                     SCREENHEIGHT)
                  (SETQ BOTTOM (- SCREENHEIGHT HEIGHT]
             (T [SETQ WHERE (COND
                                ((POSITIONP WHERE))
                                (T (GETBOXPOSITION WIDTH HEIGHT]
                (SETQ LEFT (fetch XCOORD of WHERE))
                (SETQ BOTTOM (fetch YCOORD of WHERE]
          (COND
             ((WINDOWP PROCESS.STATUS.WINDOW)
              (CLOSEW PROCESS.STATUS.WINDOW)))
          (SETQ PROCESS.STATUS.WINDOW
           (CREATEW (create REGION
                            LEFT _ LEFT
                            BOTTOM _ BOTTOM
                            WIDTH _ WIDTH
                            HEIGHT _ HEIGHT)))
          (ADDMENU PROCOPMENU PROCESS.STATUS.WINDOW '(0 . 0))
          (ADDMENU (SETQ PROCMENU PMENU)
                 PROCESS.STATUS.WINDOW
                 (create POSITION
                         XCOORD _ (IQUOTIENT (- WIDTH (fetch IMAGEWIDTH of PMENU))
                                             2)
                         YCOORD _ (+ (fetch IMAGEHEIGHT of PROCOPMENU)
                                     4)))                  ; Don't set PROCMENU globally until after old psw is closed
          [COND
             (SELECTEDPROC (COND
                              ((FMEMB SELECTEDPROC PROCS)
                               (SHADEITEM SELECTEDPROC PMENU SELECTIONSHADE))
                              (T (SETQ SELECTEDPROC]
          (WINDOWPROP PROCESS.STATUS.WINDOW 'PROCS PROCS)
          (WINDOWPROP PROCESS.STATUS.WINDOW 'MINSIZE (CONS 0 HEIGHT))
          (WINDOWPROP PROCESS.STATUS.WINDOW 'MAXSIZE (CONS MAX.SMALLP HEIGHT))
                                                           ; Window is of fixed size for attached window reshaping
          (WINDOWPROP PROCESS.STATUS.WINDOW 'CLOSEFN (FUNCTION (LAMBDA (WINDOW)
                                                                 (COND
                                                                    ((EQ WINDOW PROCESS.STATUS.WINDOW)
                                                                     (SETQ PROCMENU (SETQ PROCESS.STATUS.WINDOW]
)
```

## (\PSW.SELECTED

```
  [LAMBDA (ITEM MENU BUTTON)                               (* bvm%: " 6-JUN-82 21:03")
    (COND
       ((AND SELECTEDPROC (NEQ ITEM SELECTEDPROC))
        (SHADEITEM SELECTEDPROC MENU WHITESHADE)))
    (SHADEITEM ITEM MENU SELECTIONSHADE)
    (SETQ SELECTEDPROC ITEM])
```

## (\PSWOP.SELECTED

```
[LAMBDA (ITEM MENU BUTTON)                                    ; Edited 12-Oct-87 18:28 by bvm:
  (COND
     ((NULL (THIS.PROCESS))
      (PROMPTPRINT "Processes are off!"))
     [(EQ ITEM 'WHO?)
      (COND
         ((TTY.PROCESS)
          (\PSW.SELECTED (fetch PROCNAME of (TTY.PROCESS))
                 PROCMENU))
         (T (PROMPTPRINT "No process has the tty!!!"]
     (SELECTEDPROC
      (PROG ((P (FIND.PROCESS SELECTEDPROC))
             VALUE)
            (OR P (RETURN (PROMPTPRINT "Can't find process " SELECTEDPROC)))
            (SELECTQ ITEM
                 (KBD_ (TTY.PROCESS P))
                 ((BT BTV BTV* BTV!)
                     (PROCESS.BACKTRACE P ITEM))
                 (INFO [COND
                          ((NOT (SETQ VALUE (fetch PROCINFOHOOK of P)))
                           (PROMPTPRINT "No info program supplied for this process"))
                          ((AND (LISTP VALUE)
                                (NOT (FMEMB (CAR VALUE)
                                            LAMBDASPLST)))
                           (PROCESS.EVAL P VALUE))
                          (T (PROCESS.APPLY P VALUE (LIST P BUTTON])
                 (KILL [COND
                          ((COND
                              ((OR (fetch PROCSYSTEMP of P)
                                   (EQ (fetch PROCNAME of P)
                                       'EXEC))
                               (MOUSECONFIRM "Click LEFT to confirm killing system process" T NIL (WFROMMENU
                                                                                                   MENU)))
                              (T T))
                           (DEL.PROCESS P)
                           (forDuration 500 until (fetch PROCDELETED of P) do (BLOCK))
                           (if (EQ (WINDOWPROP PROCESS.STATUS.WINDOW 'BUTTONEVENTFN)
                                   '\UPDATE.PROCESS.WINDOW)
                               then                           ; Repaint the window after the kill
                                    (PROCESS.STATUS.WINDOW T)
                                    (ERROR!)                  ; Don't let the mouse handler think any longer about the old
                                                              ; window, lest it bring it to top to handle the selection
                                    ])
                 (RESTART (RESTART.PROCESS P))
                 (WAKE (PROG (VALUE)
                             (WAKE.PROCESS P (SELECTQ [MENU (OR PROCOP.WAKEMENU
                                                                (SETQ PROCOP.WAKEMENU
                                                                 (create MENU
                                                                         ITEMS _ '((NIL 'NULL)
                                                                                   T Other)
                                                                         TITLE _ "WakeUp Value"
                                                                         CENTERFLG _ T]
                                                        (NIL (RETURN))
                                                        (NULL NIL)
                                                        (T T)
                                                        (Other (CAR (OR (LISTP (PROCESS.READ "Value to return to
                                                                                             woken process: "))
                                                                        (RETURN))))
                                                       NIL))))
                 (BREAK (BREAK.PROCESS P))
                 (SUSPEND (AND (NEQ P (THIS.PROCESS))
                               (\SUSPEND.PROCESS P)))
                NIL])
```

## (PROCESS.BACKTRACE

```
[LAMBDA (PROC CMD WINDOW)                                     (* jds " 4-Feb-86 14:52")
  (PROG (DSP PLACE REGION)
        [COND
           ([NOT (WINDOWP (OR WINDOW (SETQ WINDOW (CAR (ATTACHEDWINDOWS PROCESS.STATUS.WINDOW]
            (SETQ REGION (WINDOWPROP PROCESS.STATUS.WINDOW 'REGION))
            (SETQ DSP (WINDOWPROP (SETQ WINDOW (CREATEW (create REGION
                                                                LEFT _ (fetch (REGION LEFT) of REGION)
                                                                BOTTOM _ (COND
                                                                            ((ILESSP (fetch (REGION BOTTOM)
                                                                                        of REGION)
                                                                                     PROCBACKTRACEHEIGHT)
                                                                             (SETQ PLACE 'TOP)
                                                                             (fetch (REGION TOP) of REGION))
                                                                            (T (SETQ PLACE 'BOTTOM)
                                                                               (IDIFFERENCE (fetch (REGION
                                                                                                    BOTTOM
                                                                                                    )
                                                                                                  of REGION)
                                                                                            PROCBACKTRACEHEIGHT)))
                                                                WIDTH _ (fetch (REGION WIDTH) of REGION)
```

```
                                                                HEIGHT _ PROCBACKTRACEHEIGHT)
                                                        "Process backtrace" NIL T))
                              'DSP))
                (ATTACHWINDOW WINDOW PROCESS.STATUS.WINDOW PLACE 'JUSTIFY 'LOCALCLOSE)
                (DSPSCROLL 'OFF DSP)
                (WINDOWPROP WINDOW 'PASSTOMAINCOMS '(MOVEW SHRINKW BURYW))
                (DSPFONT (OR BACKTRACEFONT (FONTCREATE 'GACHA 8))
                         DSP))
             (T (SETQ DSP (WINDOWPROP WINDOW 'DSP]
          (DSPRESET DSP)
          (LET ((PLVLFILEFLG T)
                (FX (fetch (PROCESS PROCFX) of PROC))
                STKP)
               (BAKTRACE [COND
                            ((EQ FX 0)                          ; The currently active proc!
                             '\PSWOP.SELECTED)
                            (T (SETQ STKP (\MAKESTACKP NIL FX]
                         NIL NIL (SELECTQ CMD
                                      (BT 0)
                                      (BTV 1)
                                      (BTV* 7)
                                      (BTV! 39)
                                      0)
                         DSP)
               (AND STKP (RELSTK STKP])
```

## (\INVALIDATE.PROCESS.WINDOW

```
  [LAMBDA NIL                                                   (* bvm%: "21-JUN-82 17:50")
    ;; If process window is active and correct, grays it out and makes its buttoneventfn be something to update it
    (PROG (OLDBUTTONFN)
          (COND
             ((AND PROCESS.STATUS.WINDOW (ACTIVEWP PROCESS.STATUS.WINDOW)
                   (NEQ (SETQ OLDBUTTONFN (WINDOWPROP PROCESS.STATUS.WINDOW 'BUTTONEVENTFN
                                                      '\UPDATE.PROCESS.WINDOW))
                        '\UPDATE.PROCESS.WINDOW))
              (WINDOWPROP PROCESS.STATUS.WINDOW 'OLDBUTTONEVENTFN OLDBUTTONFN)
              (DSPFILL NIL LIGHTGRAYSHADE 'INVERT PROCESS.STATUS.WINDOW])
```

## (\UPDATE.PROCESS.WINDOW

```
  [LAMBDA (WINDOW)                                              (* bvm%: " 4-OCT-83 11:54")
    (PROG (OLDBUTTONFN)                                         ; Restore proper button fn
          (COND
             ((for P in \PROCESSES as NAME in (WINDOWPROP WINDOW 'PROCS) thereis (NEQ NAME (fetch PROCNAME
                                                                                             of P)))
              (PROCESS.STATUS.WINDOW T))
             (T (DSPFILL NIL LIGHTGRAYSHADE 'INVERT PROCESS.STATUS.WINDOW)
                (WINDOWPROP WINDOW 'BUTTONEVENTFN (SETQ OLDBUTTONFN (WINDOWPROP WINDOW 'OLDBUTTONEVENTFN NIL)))
                                                              ; Now invoke the real fn
                (APPLY* OLDBUTTONFN WINDOW])

)

(RPAQ? PROCMENU )

(RPAQ? PROCOPMENU )

(RPAQ? PROCOP.WAKEMENU )

(RPAQ? PROCESS.STATUS.WINDOW )

(RPAQ? SELECTEDPROC )

(RPAQ? PROCBACKTRACEHEIGHT 320)

(ADDTOVAR BackgroundMenuCommands ("PSW" '(PROCESS.STATUS.WINDOW)
                                  "Puts up a Process Status Window"))

(SETQQ BackgroundMenu)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS PROCESS.STATUS.WINDOW PROCMENU PROCOPMENU PROCOP.WAKEMENU PROCBACKTRACEHEIGHT SELECTEDPROC
       BACKTRACEFONT)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ LIGHTGRAYSHADE 1)

(RPAQQ SELECTIONSHADE 520)

(CONSTANTS LIGHTGRAYSHADE SELECTIONSHADE)
```

```
)
)

(DECLARE%: DONTEVAL@LOAD DOCOPY

(ADDTOVAR WINDOWUSERFORMS (\PROC.AFTER.WINDOWWORLD))

(DEFPRINT 'PROCESS (FUNCTION \PROCESS.DEFPRINT))

(DEFPRINT 'EVENT (FUNCTION \EVENT.DEFPRINT))

(DEFPRINT 'MONITORLOCK (FUNCTION \MONITORLOCK.DEFPRINT))

;; \process.init must come last, since it does a HARDRESET

(\PROCESS.INIT)
)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA PROCESSPROP ADD.PROCESS)
)
```

## FUNCTION INDEX



## VARIABLE INDEX



## MACRO INDEX



## CONSTANT INDEX



## RECORD INDEX



## PROPERTY INDEX