```
-- DebugProcess.mesa
-- Edited by:
--            Sandman on May 25, 1978  4:56 PM
--            Barbara on June 21, 1978  9:03 AM

DIRECTORY
  ControlDefs: FROM "controldefs" USING [FrameHandle, NullFrame],
  DebugContextDefs: FROM "debugcontextdefs" USING [ResetContext],
  DebuggerDefs: FROM "debuggerdefs" USING [
    addbitaddrs, CatchFrame, DumpCatchFrame, DumpNoSymbols, FrameRelBPC,
    fullsymaddress, GetValue, NoPreviousFrame, PreviousFrame,
    SOPointer, SymbolObject, SymFrameHandle, WriteFrameLocus, WriteSource],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    ControlDEL, CopyRead, InterpretString, LookupFail,
    StringExpressionToNumber, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    GetCurrentStateFromPSB, MREAD, SREAD, ValidatePSB],
  DIActionDefs: FROM "diactiondefs" USING [CleanUp, espTosop],
  DIDefs: FROM "didefs" USING [ESPointer],
  IODefs: FROM "iodefs" USING [
    CR, DEL, ReadChar, Rubout, SP, WriteChar, WriteOctal, WriteString],
  ProcessDefs: FROM "processdefs" USING [
    CurrentPSB, Empty, MonitorLock, ProcessHandle, PSB],
  SDDefs: FROM "sddefs" USING [SD, sFirstProcess, sLastProcess],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  SymbolTableDefs: FROM "symboltabledefs" USING [NoSymbolTable],
  SymDefs: FROM "symdefs" USING [
    CBTIndex, CBTNull, CSEIndex, CTXIndex, CTXNull, ISEIndex, SENull];

DebugProcess: PROGRAM
IMPORTS DebugContextDefs, DebuggerDefs, DebugMiscDefs, DebugSymbolDefs,
  DebugUtilityDefs, DIActionDefs, IODefs, StreamDefs, SymbolTableDefs
EXPORTS DebugContextDefs
SHARES ProcessDefs =

BEGIN

PSB: TYPE = ProcessDefs.PSB;
ProcessHandle: TYPE = ProcessDefs.ProcessHandle;
SOPointer: TYPE = DebuggerDefs.SOPointer;

InvalidPSB: PUBLIC SIGNAL [psb: UNSPECIFIED] = CODE;

SetProcessContext: PUBLIC PROCEDURE [p: STRING] =
  BEGIN
  psb: ProcessHandle ← StringToPSB[p];
  DebugContextDefs.ResetContext[FrameFromPSB[psb], psb];
  RETURN
  END;

StringToPSB: PROCEDURE [p: STRING] RETURNS [psb: ProcessHandle] =
  BEGIN
  FindPSB: PROCEDURE [esp: DIDefs.ESPointer] =
    BEGIN OPEN ProcessDefs, DebuggerDefs;
    so: SymbolObject;
    sop: SOPointer ← @so;
    DIActionDefs.espTosop[esp, sop];
    psb ← LOOPHOLE[GetValue[sop], ProcessHandle];
    RETURN
    END;

  BEGIN
  IF p[0] IN ['0..'9] THEN
    psb ← LOOPHOLE[DebugMiscDefs.StringExpressionToNumber[p, 8],ProcessHandle]
  ELSE DebugMiscDefs.InterpretString[p, FindPSB, TRUE ! ANY => GOTO signalOnce];
  IF ~DebugUtilityDefs.ValidatePSB[psb] THEN SIGNAL InvalidPSB[psb];
  EXITS
    signalOnce =>
      BEGIN DIActionDefs.CleanUp[]; SIGNAL DebugMiscDefs.LookupFail[p]; END;
  END;
  RETURN
  END;

ListProcesses: PUBLIC PROCEDURE =
```

```
  BEGIN OPEN DebugUtilityDefs, SDDefs;
  fakePSB: ProcessHandle ← SREAD[@SD[sFirstProcess]];
  lastPSB: ProcessHandle ← SREAD[@SD[sLastProcess]];
  DO
    IF ValidatePSB[fakePSB] AND StateOK[fakePSB] THEN DumpPSB[fakePSB];
    IF fakePSB = lastPSB THEN EXIT;
    fakePSB ← fakePSB + SIZE[PSB];
    DebugMiscDefs.WriteEOL[];
    IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
    ENDLOOP;
  RETURN
  END;

StateOK: PROCEDURE [psb: ProcessHandle] RETURNS [BOOLEAN] =
  BEGIN
  localPSB: PSB;
  DebugMiscDefs.CopyRead[from: psb, to: @localPSB, nwords: SIZE[PSB]];
  RETURN[localPSB.state # dead]
  END;

DumpProcessStack: PROCEDURE [psb: ProcessHandle] =
  BEGIN
  c: CHARACTER;
  head: ProcessHandle ← psb;
  DO
    DebugMiscDefs.WriteEOL[];
    IODefs.WriteString["   >"L];
    c ← IODefs.ReadChar[];
    IODefs.WriteChar[c];
    SELECT c FROM
      'n, 'N  => IF (psb←GetNextPSB[psb])= head OR psb = NIL THEN EXIT
                 ELSE BEGIN DebugMiscDefs.WriteEOL[]; DumpPSB[psb]; END;
      'p, 'P  => DumpPriority[psb];
      'q, 'Q, IODefs.DEL  => EXIT;
      'r, 'R  => BEGIN DebugMiscDefs.WriteEOL[]; DumpRoot[psb]; END;
      's, 'S  => BEGIN DebugMiscDefs.WriteEOL[]; DumpSource[psb]; END;
      ENDCASE => IODefs.WriteString[" --Options are: N, P, Q, R, S"L];
    ENDLOOP;
  RETURN
  END;

GetNextPSB: PROCEDURE [psb: ProcessHandle] RETURNS [ProcessHandle] =
  BEGIN OPEN ProcessDefs, SDDefs, DebugUtilityDefs; --from PSB array
  firstPSB: ProcessHandle ← SREAD[@SD[sFirstProcess]];
  lastPSB: ProcessHandle ← SREAD[@SD[sLastProcess]];
  head: ProcessHandle ← psb;
  DO
    psb ← IF psb # lastPSB THEN psb+SIZE[PSB] ELSE firstPSB;
    IF psb = head THEN GOTO done;
    IF ValidatePSB[psb] AND StateOK[psb] THEN EXIT;
    REPEAT
      done => RETURN[NIL];
    ENDLOOP;
  RETURN[psb]
  END;

DisplayQueue: PUBLIC PROCEDURE [q: STRING] =
  BEGIN
  qHead: ProcessHandle;
  cv: BOOLEAN;
  [qHead, cv] ← StringToQueue[q];
  IF qHead = NIL OR (qHead ← StartQueue[qHead, cv]) = NIL THEN
    BEGIN IODefs.WriteString[" Queue empty!"L]; RETURN END;
  DumpPSB[qHead];
  DumpQueueStack[qHead];
  RETURN
  END;

StartQueue: PROCEDURE [psb: ProcessHandle, cv: BOOLEAN]
  RETURNS [ProcessHandle] =
  BEGIN
  cleanupLink: ProcessHandle;
  IF ~cv THEN RETURN[DebugUtilityDefs.SRFAD[@psb.link]];
  cleanupLink ← DebugUtilityDefs.SREAD[@psb.cleanup];
  IF cleanupLink = NIL THEN RETURN[DebugUtilityDefs.SREAD[@psb.link]];
  UNTIL cleanupLink = NIL OR cleanupLink = psb DO
```

```
    psb ← cleanupLink;
    cleanupLink ← DebugUtilityDefs.SREAD[@cleanupLink.cleanup];
    ENDLOOP;
  RETURN[IF cleanupLink = NIL THEN psb ELSE NIL];
  END;

DumpQueueStack: PROCEDURE [psb: ProcessHandle] =
  BEGIN
  c: CHARACTER;
  head: ProcessHandle ← psb;
  DO
    DebugMiscDefs.WriteEOL[];
    IODefs.WriteString["   >"L];
    c ← IODefs.ReadChar[];
    IODefs.WriteChar[c];
    SELECT c FROM
      'n, 'N  => IF (psb ← DebugUtilityDefs.SREAD[@psb.link])= head THEN EXIT
                 ELSE BEGIN DebugMiscDefs.WriteEOL[]; DumpPSB[psb]; END;
      'p, 'P  => DumpPriority[psb];
      'q, 'Q, IODefs.DEL  => EXIT;
      'r, 'R  => BEGIN DebugMiscDefs.WriteEOL[]; DumpRoot[psb]; END;
      ENDCASE => IODefs.WriteString[" --Options are: N, P, Q, R"L];
    ENDLOOP;
  RETURN
  END;

StringToQueue: PROCEDURE [q: STRING] RETURNS [qHead: ProcessHandle, cv: BOOLEAN] =
  BEGIN OPEN DebugUtilityDefs;
  fail: BOOLEAN ← FALSE;
  FindQueue: PROCEDURE [esp: DIDefs.ESPointer] =
    BEGIN OPEN ProcessDefs, DebuggerDefs;
    so: SymbolObject;
    sop: SOPointer ← @so;
    mLock: MonitorLock;
    PSBBase: CARDINAL = 0;
    found: BOOLEAN;
    DIActionDefs.espTosop[esp, sop];
    BEGIN OPEN sop.stbase;
    WITH (seb+UnderType[sop.tsei]) SELECT FROM
        record => IF fieldctx = MonitorLockCtxIndex THEN GOTO done;
      ENDCASE;
    [found, cv] ← SearchForMonitorLock[sop];
    IF ~found THEN GOTO fail;
    EXITS
      done => cv ← FALSE;
      fail => BEGIN fail ← TRUE; RETURN END;
    END;
    mLock ← LOOPHOLE[GetValue[sop], MonitorLock];
    qHead ← IF mLock.queue = Empty THEN NIL ELSE mLock.queue + PSBBase;
    RETURN
    END;

  BEGIN
  IF q[0] IN ['0..'9] THEN
    BEGIN
    qHead ← SREAD[DebugMiscDefs.StringExpressionToNumber[q, 8]];
    IODefs.WriteString["condition variable? [Y or N]"L];
    cv ← YesNo[];
    END
  ELSE DebugMiscDefs.InterpretString[q, FindQueue, TRUE !
        ANY => GOTO signalOnce];
  EXITS
    signalOnce => BEGIN DIActionDefs.CleanUp[]; fail ← TRUE; END;
  END;
  IF fail THEN SIGNAL DebugMiscDefs.LookupFail[q];
  RETURN
  END;

YesNo: PROCEDURE RETURNS [BOOLEAN] =
  BEGIN OPEN IODefs;
  DO
    SELECT ReadChar[] FROM
      'y,'Y,CR,SP => BEGIN WriteString[" yes"L]; RETURN[TRUE] END;
      'n,'N => BEGIN WriteString[" no"L]; RETURN[FALSE] END;
      DEL => SIGNAL Rubout;
      ENDCASE => WriteChar['?];
```

```
    ENDLOOP;
  RETURN[FALSE];
  END;

SearchForMonitorLock: PROCEDURE [sop: SOPointer] RETURNS [found, cv: BOOLEAN] =
  BEGIN OPEN DebuggerDefs, SymDefs;
  cbti: CBTIndex;
  c: CTXIndex ← CTXNull;
  WITH (sop.stbase.seb + sop.stbase.UnderType[sop.tsei]) SELECT FROM
    record => IF monitored THEN c ← fieldctx
      ELSE IF fieldctx = ConditionCtxIndex THEN
        BEGIN OPEN sop.stbase;
        sop.baddr ← addbitaddrs[fullsymaddress[sop], sop.baddr];
        sop.sei ← FirstCtxSe[fieldctx];
        sop.tsei ← (seb+sop.sei).idtype;
        RETURN[TRUE, TRUE];
        END;
    transfer => IF mode = program THEN
      BEGIN OPEN sop.stbase;
      cbti ← (seb+sop.sei).idinfo;
      IF cbti # CBTNull THEN c ← (bb+cbti).localCtx;
      END;
    ENDCASE;
  IF c = CTXNull THEN RETURN[FALSE, FALSE];
  RETURN[SearchCtxForLock[sop, c], FALSE]
  END;

MonitorLockCtxIndex: SymDefs.CTXIndex ← LOOPHOLE[8];
ConditionCtxIndex: SymDefs.CTXIndex ← LOOPHOLE[10];

SearchCtxForLock: PROCEDURE [sop: SOPointer, c: SymDefs.CTXIndex]
  RETURNS [BOOLEAN] =
  BEGIN OPEN sop.stbase;
  sei: SymDefs.ISEIndex;
  tsei: SymDefs.CSEIndex;
  FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = SymDefs.SENull DO
    --look for the type of sei = MONITORLOCK
    tsei ← UnderType[(seb+sei).idtype];
    WITH (seb+tsei) SELECT FROM
      record => IF fieldctx = MonitorLockCtxIndex THEN
        BEGIN
        sop.sei ← sei; sop.tsei ← (seb+sei).idtype;
        RETURN[TRUE]
        END;
      ENDCASE;
    ENDLOOP;
  RETURN[FALSE]
  END;

FrameFromPSB: PROCEDURE [psb: ProcessHandle] RETURNS [ControlDefs.FrameHandle] =
  --get the frame for the currently running process from the StateVector
  BEGIN OPEN DebugUtilityDefs;
  RETURN[IF psb # SREAD[ProcessDefs.CurrentPSB] THEN SREAD[@psb.frame]
         ELSE SREAD[@GetCurrentStateFromPSB[].dest]]
  END;

DisplayProcess: PUBLIC PROCEDURE [p: STRING] =
  BEGIN
  psb: ProcessHandle;
  DumpPSB[psb ← StringToPSB[p]];
  DumpProcessStack[psb];
  RETURN
  END;

DumpPSB: PROCEDURE [psb: ProcessHandle] =
  BEGIN OPEN DebuggerDefs;
  frame: ControlDefs.FrameHandle;
  IF ~DebugUtilityDefs.ValidatePSB[psb] THEN SIGNAL InvalidPSB[psb];
  DebugMiscDefs.WriteEOL[];
  IODefs.WriteString["PSB: "L]; IODefs.WriteOctal[psb];
  IODefs.WriteString[", "L];
  DumpWaiting[psb];
  IF (frame ← FrameFromPSB[psb]) = ControlDefs.NullFrame THEN RETURN;
  IF CatchFrame[frame] THEN DumpCatchFrame[frame]
    ELSE ShowFrame[frame];
  RETURN
```

```
  END;

ShowFrame: PROCEDURE [frame: ControlDefs.FrameHandle]=
  BEGIN OPEN DebuggerDefs, DebugSymbolDefs;
  f: SymFrameHandle;
  BEGIN
  f.stbase ← DAcquireSymbolTable[SymbolsForFrame[frame !
      SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nosym] !
    SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nosym];
  f.faddr ← frame;
  WriteFrameLocus[f, FALSE];
  DReleaseSymbolTable[f.stbase];
  EXITS
    nosym => DumpNoSymbols[frame];
  END;
  RETURN
  END;

DumpSource: PROCEDURE [psb: ProcessHandle]=
  BEGIN OPEN DebuggerDefs, DebugUtilityDefs;
  frame: ControlDefs.FrameHandle ← FrameFromPSB[psb];
  WriteSource[MREAD[@frame.accesslink], FrameRelBPC[frame], TRUE !
    SymbolTableDefs.NoSymbolTable--[seg]-- =>
      BEGIN IODefs.WriteString[" No symbol table."L]; CONTINUE END];
  RETURN
  END;

DumpRoot: PROCEDURE [psb: ProcessHandle] =
  BEGIN OPEN DebuggerDefs;
  f: ControlDefs.FrameHandle ← FrameFromPSB[psb];
  DO
    f ← PreviousFrame[f ! NoPreviousFrame => EXIT];
    ENDLOOP;
  IF f = ControlDefs.NullFrame THEN RETURN;
  IF CatchFrame[f] THEN DumpCatchFrame[f] ELSE ShowFrame[f];
  RETURN
  END;

DumpWaiting: PROCEDURE [psb: ProcessHandle] =
  BEGIN OPEN IODefs;
  localPSB: PSB;
  DebugMiscDefs.CopyRead[from: psb, to: @localPSB, nwords: SIZE[PSB]];
  IF localPSB.enterFailed THEN WriteString["waiting ML, "L]
  ELSE IF localPSB.waitingOnCV THEN WriteString["waiting CV, "L];
  RETURN
  END;

DumpPriority: PROCEDURE [psb: ProcessHandle] =
  BEGIN
  localPSB: PSB;
  DebugMiscDefs.CopyRead[from: psb, to: @localPSB, nwords: SIZE[PSB]];
  IODefs.WriteString["riority "L];
  IODefs.WriteOctal[localPSB.priority];
  RETURN
  END;

END..
```