```
-- File: Bootmesa.Mesa
-- Last edited by Sandman; May 24, 1978  11:53 AM

DIRECTORY
  AllocDefs: FROM "allocdefs",
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  BootCacheDefs: FROM "bootcachedefs",
  BootmesaDefs: FROM "bootmesadefs",
  CommanderDefs: FROM "commanderdefs",
  ControlDefs: FROM "controldefs",
  FakeSegDefs: FROM "fakesegdefs",
  FileLookupDefs: FROM "filelookupdefs",
  ImageDefs: FROM "imagedefs",
  IODefs: FROM "iodefs",
  InlineDefs: FROM "inlinedefs",
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs",
  LoadStateDefs: FROM "loadstatedefs",
  MiscDefs: FROM "miscdefs",
  OsStaticDefs: FROM "osstaticdefs",
  ProcessDefs: FROM "processdefs",
  SDDefs: FROM "sddefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs",
  TimeDefs: FROM "timedefs",
  WartDefs: FROM "wartdefs";

DEFINITIONS FROM FakeSegDefs, AltoDefs, ControlDefs, WartDefs, BootmesaDefs;

Bootmesa: PROGRAM [data: POINTER TO BootData]
  IMPORTS BootCacheDefs, BootmesaDefs, CommanderDefs, IODefs, MiscDefs,
    SegmentDefs, StringDefs, TimeDefs, FakeSegDefs, LoaderBcdUtilDefs
  EXPORTS BootmesaDefs, FakeSegDefs
  SHARES ControlDefs, ImageDefs, ProcessDefs =
BEGIN

FileHandle: TYPE = SegmentDefs.FileHandle;
FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
DataSegmentHandle: TYPE = SegmentDefs.DataSegmentHandle;


-- utility procedures

BootAbort: PUBLIC SIGNAL = CODE;

SwapInFrame: PUBLIC PROCEDURE[f: GlobalFrameHandle] =
  BEGIN
  codeseg: FakeSegmentHandle ← BootCacheDefs.READ[@f.codesegment];
  FakeSegDefs.FakeSwapIn[codeseg];
  END;

BootmesaError:  PUBLIC PROCEDURE [msg: STRING] =
  BEGIN OPEN IODefs;
  WriteString["Bootmesa Error"L];
  WriteLine[msg];
  SIGNAL BootAbort;
  END;


-- Frame Allocation

paramsperframe: CARDINAL = 1;
stringsizeperframe: CARDINAL = 10;
framesloaded: CARDINAL ← 0;
framebase: POINTER;
FrameStackFull: ERROR = CODE;
extraFramesDesired: BOOLEAN ← FALSE;

framevec: PACKED ARRAY [0..18] OF CARDINAL =
  [7,11,15,19,23,27,31,39,47,55,67,79,95,111,127,147,171,199,231];
frameweight: PACKED ARRAY [0..19] OF CARDINAL ←
  [9,13,9,8,7,6,4,2,2,1,1,1,1,1,1,1,1,1,1,0];
extraframes: ARRAY [0..MaxAllocSlot) OF CARDINAL;
```

```
AllocGlobalFrame: PUBLIC PROCEDURE [
  framesize, nlinks: CARDINAL, framelinks: BOOLEAN] RETURNS [frame: POINTER] =
  BEGIN
  framebase ← framebase - framesize;
  frame ← framebase ← framebase - LOOPHOLE[framebase,CARDINAL] MOD 4;
  IF framelinks THEN framebase ← framebase - nlinks;
  IF LOOPHOLE[framebase,CARDINAL] < CARDINAL[FirstVMPage*PageSize] THEN
    ERROR FrameStackFull;
  RETURN
  END;

AllocFrame: PROCEDURE [findex: CARDINAL] RETURNS [frame: POINTER] =
  BEGIN
  framesloaded ← framesloaded + 1;
  frame ← framebase ← framebase - (framevec[findex]+1);
  BootCacheDefs.WRITE[frame-1, findex];
  RETURN
  END;

AddFrame: PROCEDURE [findex: CARDINAL] =
  BEGIN OPEN BootCacheDefs;
  p: POINTER = AllocFrame[findex];
  WRITE[p, READ[AVbase+findex]];
  WRITE[AVbase+findex, p];
  END;

GetExtraFrames: PUBLIC PROCEDURE =
  BEGIN
  extraFramesDesired ← TRUE;
  END;

SetExtraFrames: PUBLIC PROCEDURE =
  BEGIN OPEN IODefs;
  i: CARDINAL;
  IF ~extraFramesDesired THEN RETURN;
  WriteLine["Index Size Extra"L];
  FOR i IN [0..LENGTH[extraframes]) DO
    WriteNumber[i,NumberFormat[10,FALSE,FALSE,5]];
    WriteNumber[framevec[i],NumberFormat[10,FALSE,FALSE,5]];
    WriteChar[' ];
    extraframes[i] ← ReadNumber[0,10 !
      StringDefs.InvalidNumber, Rubout, LineOverflow =>
        BEGIN WriteString[" ? "L]; RETRY END];
    WriteChar[CR];
    ENDLOOP;
  END;

InitializeExtraFrames: PROCEDURE =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..LENGTH[extraframes]) DO
    extraframes[i] ← 0;
    ENDLOOP;
  extraFramesDesired ← FALSE;
  END;

AllocateExtraFrames: PROCEDURE =
  BEGIN
  i,j: CARDINAL;
  FOR i IN [0..LENGTH[extraframes]) DO
    FOR j IN [0..extraframes[i]) DO
      AddFrame[i];
      ENDLOOP;
    ENDLOOP;
  END;

DivideAllocationArea: PROCEDURE [base: Address, size: CARDINAL] =
  BEGIN OPEN BootCacheDefs;
  p: Address ← base+4;  -- allow initial dead space
  i, j, s, c, sum: CARDINAL;

  fw: ARRAY [0..LENGTH[frameweight]) OF CARDINAL;

  FOR i IN [0..LENGTH[fw]) DO fw[i]←frameweight[i] ENDLOOP;
  size ← size-4;
```

```
  sum←0;
  FOR i IN [0..LENGTH[fw]) DO sum ← sum + framevec[i]*fw[i] ENDLOOP;
  c ← MAX[size/sum,1];

  WHILE size > framevec[0] DO
    FOR i IN[0..LENGTH[framevec]) DO
      s ← framevec[i]+1;
      FOR j IN[1..fw[i]*c] DO
        -- add to alloc vector
        IF size < s THEN EXIT;
        WRITE[p-1,i];     -- hidden link word
        WRITE[p, READ[AVbase+i]];
        WRITE[AVbase+i, p];
        p←p+s; size←size-s;
        ENDLOOP;
      ENDLOOP;
    FOR i IN[0..LENGTH[fw]-1) DO fw[i]←MAX[fw[i]/3,1] ENDLOOP;
    c←1;
    ENDLOOP;
  RETURN
  END;


-- initialization

AVbase, GFTbase, SDbase, SVbase: POINTER ← NIL;

VMFileHandle: FileHandle;
DefaultFirstVMPage: PageNumber ← 2;
DefaultLastVMPage: PageNumber ← 370B;
FirstVMPage: PageNumber ← 0;
LastVMPage: PageNumber ← 0;

SetMemoryLimits: PROCEDURE [fp, lp: PageNumber] =
  BEGIN FirstVMPage ← fp; LastVMPage ← lp; END;

SetDefaultMemoryLimits: PUBLIC PROCEDURE [fp, lp: PageNumber] =
  BEGIN DefaultFirstVMPage ← fp; DefaultLastVMPage ← lp; END;

StateVectors: CARDINAL =
  (LAST[ProcessDefs.Priority]+1)*SIZE[ControlDefs.StateVector];

DefaultNProcesses: PUBLIC CARDINAL ←
  (AltoDefs.PageSize-StateVectors)/SIZE[ProcessDefs.PSB];

nProcesses: CARDINAL ← 0;

SetNumberProcesses: PROCEDURE [n: CARDINAL] = BEGIN nProcesses ← n; END;

SetDefaultNProcesses: PUBLIC PROCEDURE [n: CARDINAL] =
  BEGIN DefaultNProcesses ← n; END;

-- ************************************

  AVregister: POINTER = LOOPHOLE[1000B];
  GFTregister: POINTER = LOOPHOLE[1400B];
  SDoffset: CARDINAL = 60B;
-- ************************************

FramePages: PageCount;

DefaultGFTLength: CARDINAL ← 256;
GFTLength: CARDINAL ← 0;

SetGFTLength: PROCEDURE [l: CARDINAL] = BEGIN GFTLength ← l; END;

SetDefaultGFTLength: PUBLIC PROCEDURE [l: CARDINAL] =
  BEGIN DefaultGFTLength ← l; END;

AssignDefaults: PROCEDURE =
  BEGIN
  IF FirstVMPage = 0 THEN FirstVMPage ← DefaultFirstVMPage;
  IF LastVMPage = 0 THEN LastVMPage ← DefaultLastVMPage;
  IF GFTLength = 0 THEN GFTLength ← DefaultGFTLength;
  IF nProcesses = 0 THEN nProcesses ← DefaultNProcesses;
  RETURN
```

```
  END;

InitializeVM: PROCEDURE [framep: PageCount] =
  BEGIN OPEN SegmentDefs, BootCacheDefs, FakeSegDefs;
  s: FakeSegmentHandle;
  i, GFTpages, SVpages: CARDINAL;
  fp: PageNumber;

  AssignDefaults[];
  fp ← FirstVMPage;
  FramePages ← framep;
  FakeInitSegMachinery[FirstVMPage, LastVMPage];
  BootCacheDefs.InitCoreCache["BootMesa.Scratch", FirstVMPage, LastVMPage];
  framebase ← LOOPHOLE[(LastVMPage+1) * PageSize];
  -- av and sd
  s ← FakeNewSegment[DefaultFile,FirstVMPage,1,Read+Write];
  VMFileHandle ← s.File;
  AVbase ← LOOPHOLE[s.VMaddress];
  IF AVbase # AVregister THEN BootmesaError["Invalid AV"L];
  SDbase ← AVbase + SDoffset;
  FOR i IN[0..PageSize) DO WRITE[AVbase+i,0] ENDLOOP;
  FOR i IN[0..MaxAllocSlot) DO WRITE[AVbase+i, 4*(i+1)+2] ENDLOOP;
  WRITE[AVbase+11, 1];
  FOR i IN [MaxAllocSlot..MaxAllocSlot+2] DO WRITE[AVbase+i, 1] ENDLOOP;
  --gft
  fp ← fp+1;
  GFTpages ← (GFTLength*SIZE[GFTItem]+(PageSize-1))/PageSize;
  s ← FakeNewSegment[DefaultFile,fp,GFTpages,Read+Write];
  GFTbase ← LOOPHOLE[s.VMaddress];
  IF GFTbase # GFTregister THEN BootmesaError["Invalid GFT"L];
  FOR i IN[0..GFTLength*SIZE[GFTItem]) DO WRITE[GFTbase+i,0] ENDLOOP;
  BootmesaDefs.InitializeGFT[GFTbase, GFTLength];
  WRITE[SDbase+SDDefs.sGFTLength, GFTLength];
  -- process storage initialization
  SVpages ← (nProcesses*SIZE[ProcessDefs.PSB] + StateVectors
    + (PageSize-1))/PageSize;
  s ← FakeNewSegment[DefaultFile, fp+GFTpages, SVpages, Read+Write];
  SVbase ← LOOPHOLE[s.VMaddress];
  WRITE[SDbase+SDDefs.sFirstStateVector, SVbase];
  WRITE[SDbase+SDDefs.sFirstProcess, SVbase+StateVectors];
  WRITE[SDbase+SDDefs.sLastProcess,
    SVbase+StateVectors+(nProcesses-1)*SIZE[ProcessDefs.PSB]];
  FOR i IN [0..SVpages*AltoDefs.PageSize) DO WRITE[SVbase+i, 0] ENDLOOP;
  data.AVbase ← AVbase;
  data.GFTbase ← GFTbase;
  data.SDbase ← SDbase;
  RETURN
  END;

InitializeHeap: PUBLIC PROCEDURE =
  BEGIN
  stackbase: PageNumber;
  stackpages: PageCount;
  frameseg: FakeSegmentHandle;

  AllocateExtraFrames[];
  stackbase ← LOOPHOLE[framebase,CARDINAL]/PageSize;
  stackpages ← LastVMPage-stackbase+1;
  frameseg ←
    FakeNewSegment[DefaultFile, stackbase-FramePages, FramePages+stackpages,
      SegmentDefs.Read+SegmentDefs.Write];

  DivideAllocationArea[frameseg.VMaddress, LOOPHOLE[framebase-frameseg.VMaddress]];
  RETURN
  END;

imageStamp: BcdDefs.VersionStamp;

InitializeBootmesa: PUBLIC PROCEDURE [framep: PageCount, root: STRING]
  RETURNS [BcdDefs.VersionStamp] =
  BEGIN
  InitializeVM[framep];
  InitializeBootScript[];
  InitializeExtraFrames[];
  data.imageFileRoot.length ← 0;
  StringDefs.AppendString[data.imageFileRoot, root];
```

```
  BootmesaDefs.OpenLoadmap[root];
  imageStamp ← [zapped:FALSE, net: MiscDefs.GetNetworkNumber[],
    host: OsStaticDefs.OsStatics.SerialNumber,
    time: TimeDefs.CurrentDayTime[]];
  RETURN[imageStamp]
  END;

AdjustBcd: PUBLIC PROCEDURE [fs: FakeSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  s: FileSegmentHandle;
  bcd: POINTER TO BcdDefs.BCD;
  sgb: CARDINAL;
  AdjustCodeSegment: PROCEDURE [mth: BcdDefs.MTHandle, mti: BcdDefs.MTIndex]
    RETURNS [BOOLEAN] =
    BEGIN
    frame: GlobalFrameHandle ←
      BootCacheDefs.READ[@ControlDefs.GFT[mth.gfi].frame];
    (sgb+mth.code.sgi).file ← BcdDefs.FTSelf;
    (sgb+mth.code.sgi).base ← LOOPHOLE[
      BootCacheDefs.READ[@frame.codesegment], FakeSegmentHandle].ImageBase;
    RETURN[FALSE];
    END;
  IF fs.File # data.imageFile THEN ERROR;
  s ← NewFileSegment[data.imageFile, fs.ImageBase, fs.Pages, Read+Write];
  SwapIn[s];
  bcd ← FileSegmentAddress[s];
  sgb ← LOOPHOLE[bcd+bcd.sgOffset];
  [] ← LoaderBcdUtilDefs.EnumerateModuleTable[bcd, AdjustCodeSegment];
  Unlock[s];
  DeleteFileSegment[s]; -- Swaps out first
  RETURN
  END;

DeclareLoadStateParameters: PUBLIC PROCEDURE [
  lsseg, initlsseg, bcdseg: FakeSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  gft: POINTER TO ARRAY [0..0) OF GFTItem = GFTbase;
  i: CARDINAL;
  NullFP: AltoFileDefs.FP = [[1,0,1,17777B,177777B], AltoFileDefs.eofDA];
  seg: FileSegmentHandle ← NewFileSegment[
    initlsseg.File, initlsseg.ImageBase, initlsseg.Pages, Read+Write];
  bseg: FileSegmentHandle ← NewFileSegment[
    bcdseg.File, bcdseg.ImageBase, bcdseg.Pages, Read+Write];
  loadstate: LoadStateDefs.LoadState;
  bcd: POINTER TO BcdDefs.BCD;
  SwapIn[seg];
  loadstate ← FileSegmentAddress[seg];
  MiscDefs.Zero[loadstate, AltoDefs.PageSize*seg.pages];
  SwapIn[bseg];
  bcd ← FileSegmentAddress[bseg];
  FOR i IN [0..GFTLength) DO
    IF BootCacheDefs.READ[@gft[i].frame] = ControlDefs.NullGlobalFrame THEN
      loadstate.gft[i] ← [config: LoadStateDefs.ConfigNull, gfi: 0]
      ELSE loadstate.gft[i] ← [config: 0, gfi: i];
    ENDLOOP;
  loadstate.bcds[0] ← [fp: NullFP, da: AltoFileDefs.eofDA, base: bseg.base,
    unresolved: bcd.nImports # 0, exports: bcd.nExports # 0, fill: 0,
    pages: bseg.pages];
  Unlock[seg];
  DeleteFileSegment[seg];
  Unlock[bseg];
  DeleteFileSegment[bseg];
  seg ← NewFileSegment[lsseg.File, lsseg.ImageBase, lsseg.Pages, Read+Write];
  SwapIn[seg];
  MiscDefs.Zero[FileSegmentAddress[seg], AltoDefs.PageSize*seg.pages];
  Unlock[seg];
  DeleteFileSegment[seg];
  data.image.prefix.loadStateBase ← lsseg.ImageBase;
  data.image.prefix.initialLoadStateBase ← initlsseg.ImageBase;
  data.image.prefix.loadStatePages ← lsseg.Pages;
  data.bsheader.loadState ← FakeSegDefs.GetSegmentBootLink[lsseg];
  data.bsheader.initLoadState ← FakeSegDefs.GetSegmentBootLink[initlsseg];
  data.bsheader.bcd ← FakeSegDefs.GetSegmentBootLink[bcdseg];
  END;

WriteImage: PUBLIC PROCEDURE =
```

```
  BEGIN OPEN FakeSegDefs, BootCacheDefs;
  [] ← FakeEnumerateSegments[WriteSwappedIn];
  [] ← FakeEnumerateSegments[WriteSwappedOut];

  SegmentDefs.UnlockFile[GetCoreFile[]];
  [] ← FlushCoreCache[0, AllocDefs.DefaultDataSegmentInfo, NIL];
  SetCoreFile[data.imageFile];
  data.imageStream.destroy[data.imageStream];

  RETURN
  END;

XFER: PUBLIC PROCEDURE [dest: GlobalFrameHandle] =
  BEGIN OPEN SegmentDefs;
  data.image.prefix.state.stk[0] ←
    FinishBootScript[ControlDefs.NullGlobalFrame];
  data.image.prefix.state.stkptr ← 1;
  data.image.prefix.state.dest ← SetupMainBodyFrame[dest];
  data.image.prefix.state.source ← ControlDefs.NullFrame;
  data.image.prefix.version ← imageStamp;
  BootmesaDefs.CloseLoadmap[];
  [] ← BootCacheDefs.FlushCoreCache[0, AllocDefs.DefaultDataSegmentInfo, NIL];
  Unlock[data.headerSeg]; DeleteFileSegment[data.headerSeg];
  RETURN
  END;

SetupMainBodyFrame: PROCEDURE [dest: GlobalFrameHandle]
  RETURNS [frame: FrameHandle] =
  BEGIN OPEN BootCacheDefs, SegmentDefs;
  framesize: CARDINAL;
  av: POINTER TO ARRAY [0..0) OF POINTER = AVbase;
  cseg: FakeSegmentHandle ← READ[@dest.codesegment];
  codeseg: FileSegmentHandle ←
    NewFileSegment[cseg.File, cseg.ImageBase, cseg.Pages, Read];
  code: POINTER TO CSegPrefix;
  fw: FirstFrameWord ← READ[dest];
  IF ~cseg.SwappedIn THEN BootmesaError["Dest Not Swapped In!"L];
  SwapIn[codeseg];
  code ← FileSegmentAddress[codeseg];
  code ← code + READ[@dest.code] - cseg.VMaddress;
  framesize ← code.entry[MainBodyIndex].framesize;
  IF framesize >= MaxAllocSlot THEN BootmesaError["Large Main Body Frame!"L];
  frame ← READ[@av[framesize]];
  WRITE[@av[framesize], READ[frame]];
  WRITE[@frame.pc, code.entry[MainBodyIndex].initialpc];
  WRITE[@frame.accesslink, dest];
  WRITE[@frame.returnlink, ControlDefs.NullFrame];
  fw.started ← TRUE;
  WRITE[dest, fw];
  WRITE[SDbase+SDDefs.sXferTrap, frame];
  Unlock[codeseg]; DeleteFileSegment[codeseg];
  RETURN
  END;

command: CommanderDefs.CommandBlockHandle;

  BEGIN OPEN CommanderDefs;

  command ← AddCommand["SetGFTLength", LOOPHOLE[SetGFTLength], 1];
  command.params[0] ← [type: numeric, prompt: "GFTLength"];

  command ← AddCommand["SetNumberProcesses", LOOPHOLE[SetNumberProcesses], 1];
  command.params[0] ← [type: numeric, prompt: "NumberProcesses"];

  command ← AddCommand["SetMemoryBounds", LOOPHOLE[SetMemoryLimits], 2];
  command.params[0] ← [type: numeric, prompt: "FirstVMPage"];
  command.params[1] ← [type: numeric, prompt: "LastVMPage"];

  [] ← AddCommand["GetExtraFrames",LOOPHOLE[GetExtraFrames],0];

  END;

END...
```