```
-- File: BootCache.Mesa
-- Last edited by Sandman; May 10, 1978  9:27 AM

DIRECTORY
  AllocDefs: FROM "allocdefs",
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BootCacheDefs: FROM "bootcachedefs",
  CommanderDefs: FROM "commanderdefs",
  InlineDefs: FROM "inlinedefs",
  SegmentDefs: FROM "segmentdefs",
  SystemDefs: FROM "systemdefs",
  WartDefs: FROM "wartdefs";

BootCache: PROGRAM
  IMPORTS AllocDefs, CommanderDefs, SegmentDefs, SystemDefs
  EXPORTS BootCacheDefs =
  BEGIN OPEN BootCacheDefs;

  SwapStrategy: TYPE = AllocDefs.SwapStrategy;
  PageNumber: TYPE = AltoDefs.PageNumber;
  FileHandle: TYPE = SegmentDefs.FileHandle;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  PageSize: CARDINAL = AltoDefs.PageSize;

  DefaultCacheSize: CARDINAL ← 50;      -- number of pages to keep in core
  maxsegments: CARDINAL ← 0;    -- number of pages to keep in core

  CS: DESCRIPTOR FOR ARRAY OF CoreSegmentObject;

  CoreFile: SegmentDefs.FileHandle;
  PageList: DESCRIPTOR FOR ARRAY OF PageItem;
  Fpage: PageNumber;    -- last page allocated in file
  CacheSwap: SwapStrategy ← SwapStrategy[link:,proc:AllocDefs.CantSwap];
  CodeSwap: SwapStrategy ← SwapStrategy[link:,proc:AllocDefs.TryCodeSwapping];
  FirstPage, LastPage: PageNumber;
  CreationAllowed: BOOLEAN ← TRUE;

  InitCoreCache: PUBLIC PROCEDURE [name: STRING, FirstVMPage, LastVMPage: PageNumber] =
    BEGIN OPEN SystemDefs, SegmentDefs;
    i: CARDINAL;
    IF maxsegments = 0 THEN SetCacheSize[DefaultCacheSize];
    PageList ← DESCRIPTOR[
      AllocateResidentSegment[(LastVMPage+1)*SIZE[PageItem]],
      LastVMPage+1 --, PageItem--];
    FOR i IN [0..LastVMPage] DO PageList[i].page ← 0 ENDLOOP;
    CoreFile ← NewFile[name,Read+Write+Append, DefaultVersion];
    LockFile[CoreFile];
    Fpage ← 0;
    AllocDefs.AddSwapStrategy[@CacheSwap];
    AllocDefs.AddSwapStrategy[@CodeSwap];
    FirstPage ← FirstVMPage;
    LastPage ← LastVMPage;
    CreationAllowed ← TRUE;
    RETURN
    END;


  CloseCoreCache: PUBLIC PROCEDURE =
    BEGIN OPEN SegmentDefs;
    SystemDefs.FreeSegment[BASE[CS]];
    SystemDefs.FreeSegment[BASE[PageList]];
    IF CoreFile.lock # 0 THEN UnlockFile[CoreFile];
    IF CoreFile.lock = 0 THEN ReleaseFile[CoreFile];
    AllocDefs.RemoveSwapStrategy[@CacheSwap];
    AllocDefs.RemoveSwapStrategy[@CodeSwap];
    maxsegments ← 0;
    RETURN
    END;


  FlushCoreCache: PUBLIC AllocDefs.SwappingProcedure =
    BEGIN
    did: BOOLEAN ← FALSE;
    i: CARDINAL;
    cs: CoreSegment;
```

```
    CacheSwap.proc ← AllocDefs.CantSwap;
    FOR i IN [0..maxsegments) DO
        cs←@CS[i];
        IF cs.segment # NIL THEN
          BEGIN OPEN SegmentDefs;
          did ← TRUE;
          Unlock[cs.segment];
          DeleteFileSegment[cs.segment];
          cs.segment←NIL;
          END;
        ENDLOOP;
    RETURN[did];
    END;


  NewCoreSegment: PUBLIC PROCEDURE [p: POINTER TO PageItem, cs: CoreSegment] =
    BEGIN OPEN SegmentDefs;
    s: FileSegmentHandle ← NewFileSegment[CoreFile, p.page, 1, Read];
    SetFileSegmentDA[s, p.da];
    SwapIn[s !
      SegmentFault =>
        BEGIN
        SetEndOfFile[CoreFile, p.page, AltoDefs.BytesPerPage];
        RETRY;
        END];
    p.da ← GetFileSegmentDA[s];
    cs.page ← p.page;
    cs.segment ← s;
    RETURN
    END;

  CSrover: CARDINAL ← 0;

  GetCS: PUBLIC PROCEDURE [p: PageItem] RETURNS [FileSegmentHandle] =
    BEGIN
    i: CARDINAL;
    s: FileSegmentHandle;
    sp: CoreSegment;
    BEGIN
    FOR i IN [0..maxsegments) DO
      sp ← @CS[i];
      IF sp.segment = NIL THEN GOTO newseg;
      IF sp.page = p.page THEN EXIT;
      REPEAT FINISHED =>
        BEGIN OPEN SegmentDefs;
        WHILE ~(sp←@CS[CSrover]).old DO
          sp.old ← TRUE;
          CSrover ← (CSrover+1) MOD maxsegments;
          ENDLOOP;
        CSrover ← (CSrover+1) MOD maxsegments;
        s ← sp.segment;
        sp.segment ← NIL;
        Unlock[s];
        DeleteFileSegment[s];
        GOTO newseg;
        END
    ENDLOOP;
    EXITS newseg =>
      BEGIN
      cso: CoreSegmentObject;
      NewCoreSegment[@p,@cso];
      FOR i IN [0..maxsegments) DO
        sp ← @CS[i];
        IF sp.segment = NIL THEN BEGIN sp↑ ← cso; EXIT END;
        REPEAT FINISHED => ERROR
        ENDLOOP;
      END;
    END;
    sp.old ← FALSE;
    CacheSwap.proc ← FlushCoreCache;
    RETURN [sp.segment]
    END;

  BadReadWrite: PUBLIC ERROR = CODE;

  CheckPage: PROCEDURE [cp: PageNumber] =
```

```
    BEGIN
    IF PageList[cp].page # 0 THEN RETURN;
    IF ~CreationAllowed THEN ERROR BadReadWrite;
    PageList[cp] ← PageItem[page: Fpage ← Fpage+1, da: AltoFileDefs.eofDA];
    RETURN
    END;

  READ: PUBLIC PROCEDURE [a: UNSPECIFIED] RETURNS [UNSPECIFIED] =
    BEGIN OPEN InlineDefs;
      cp: PageNumber;

      cp ← LOOPHOLE[a,CARDINAL]/PageSize;
      IF cp ~IN[FirstPage..LastPage] THEN ERROR BadReadWrite;
      CheckPage[cp];
      RETURN [(SegmentDefs.FileSegmentAddress[GetCS[PageList[cp]]] +
        BITAND[a,PageSize-1])↑];
    END;


  WRITE: PUBLIC PROCEDURE [a: UNSPECIFIED, v: UNSPECIFIED] =
    BEGIN OPEN InlineDefs;
      cp: PageNumber;
      s: SegmentDefs.FileSegmentHandle;

      cp ← LOOPHOLE[a,CARDINAL]/PageSize;
      IF cp ~IN[FirstPage..LastPage] THEN ERROR BadReadWrite;
      CheckPage[cp];
      (SegmentDefs.FileSegmentAddress[s←GetCS[PageList[cp]]] +
        BITAND[a,PageSize-1])↑ ← v;
      s.write ← TRUE;
      RETURN
    END;

  GetPageItem: PUBLIC PROCEDURE [page: CARDINAL] RETURNS [p: PageItem] =
    BEGIN
    p ← PageList[page];
    RETURN
    END;

  SetPageItem: PUBLIC PROCEDURE [page: CARDINAL, p: PageItem] =
    BEGIN
    PageList[page] ← p;
    RETURN
    END;

  SetCoreFile: PUBLIC PROCEDURE [file: FileHandle] =
    BEGIN
    CoreFile ← file;
    CreationAllowed ← FALSE;
    RETURN
    END;

  GetCoreFile: PUBLIC PROCEDURE RETURNS [FileHandle] =
    BEGIN
    RETURN[CoreFile]
    END;

  SetCacheSize: PUBLIC PROCEDURE [size: CARDINAL] =
    BEGIN OPEN SystemDefs;
    i: CARDINAL;
    maxsegments ← size;
    CS ← DESCRIPTOR[
      AllocateResidentSegment[maxsegments*SIZE[CoreSegmentObject]],
      maxsegments];
    FOR i IN [0..maxsegments) DO CS[i].segment←NIL ENDLOOP;
    END;

  SetDefaultCacheSize: PUBLIC PROCEDURE [size: CARDINAL] =
    BEGIN
    DefaultCacheSize ← size;
    END;

  -- MAIN BODY

  command: CommanderDefs.CommandBlockHandle;
```

```
  command ← CommanderDefs.AddCommand["SetCacheSize", LOOPHOLE[SetCacheSize], 1];
  command.params[0] ← [type: numeric, prompt: "Cache Size"];
  END.
```