```
-- File: ImageCache.Mesa
-- Last edited by
--                Sandman; May 18, 1978  9:56 AM

DIRECTORY
  AllocDefs: FROM "allocdefs",
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  CoreSwapDefs: FROM "coreswapdefs",
  BFSDefs: FROM "bfsdefs",
  DiskDefs: FROM "diskdefs",
  ImageDefs: FROM "imagedefs",
  ImageFileInfoDefs: FROM "imagefileinfodefs",
  InlineDefs: FROM "inlinedefs",
  SegmentDefs: FROM "segmentdefs",
  SystemDefs: FROM "systemdefs";

ImageCache: PROGRAM
  IMPORTS AllocDefs, BFSDefs, SystemDefs, SegmentDefs
  EXPORTS ImageFileInfoDefs
  SHARES ImageDefs, SegmentDefs =

BEGIN

-- User's core image management

CoreSegmentObject: TYPE = RECORD [
  segment: SegmentDefs.FileSegmentHandle,
  address: POINTER,
  lastused: CARDINAL,
  page: AltoDefs.PageNumber];

CoreSegment: TYPE = POINTER TO CoreSegmentObject;

maxsegments: CARDINAL = 8;      -- number of pages to keep in core

CS: ARRAY [0..maxsegments) OF CoreSegmentObject;

CurrentUseValue: CARDINAL;
CoreFile: SegmentDefs.FileHandle;
DAs: ARRAY [-1..256] OF AltoFileDefs.vDA;
CacheSwap: AllocDefs.SwapStrategy ←
  AllocDefs.SwapStrategy[link:,proc:AllocDefs.CantSwap];
PageMap: PACKED ARRAY [0..256) OF [0..256);

InvalidPage: PUBLIC SIGNAL [page: AltoDefs.PageNumber] = CODE;

InitImageCache: PROCEDURE [file: SegmentDefs.FileHandle] =
  BEGIN
  i: CARDINAL;
  CurrentUseValue ← 0;
  FOR i IN [0..maxsegments) DO CS[i].segment←NIL ENDLOOP;
  CoreFile ← file;
  SegmentDefs.LockFile[CoreFile];
  AllocDefs.AddSwapStrategy[@CacheSwap];
  END;

FlushCoreCache: PUBLIC AllocDefs.SwappingProcedure =
  BEGIN OPEN SegmentDefs;
  did: BOOLEAN ← FALSE;
  i: CARDINAL ← 0;
  cs: CoreSegment;
  CacheSwap.proc ← AllocDefs.CantSwap;
  FOR i IN [0..maxsegments) DO
    cs←@CS[i];
    IF cs.segment # NIL THEN
      BEGIN
      Unlock[cs.segment];
      DeleteFileSegment[cs.segment];
      cs.segment←NIL;
      did ← TRUE;
      END;
    ENDLOOP;
  CurrentUseValue←0;
  RETURN[did]
  END;
```

```
NewCoreSegment: PROCEDURE [p: AltoDefs.PageNumber, cs: CoreSegment] =
  BEGIN OPEN SegmentDefs;
  seg: FileSegmentHandle;
  seg ← NewFileSegment[CoreFile, PageMap[p], 1, Read];
  SetFileSegmentDA[seg, DAs[p]];
  SwapIn[seg];
  DAs[p] ← GetFileSegmentDA[seg];
  cs.segment ← seg;
  cs.address ← FileSegmentAddress[seg];
  cs.page ← p;
  END;

GetCS: PROCEDURE [p: AltoDefs.PageNumber] RETURNS [sp: CoreSegment] =
  BEGIN
  minUseVal: CARDINAL ← CurrentUseValue;
  minUseIndex: CARDINAL ← 0;
  i: CARDINAL;

  BEGIN
  FOR i IN [0..maxsegments) DO
    sp ← @CS[i];
    IF sp.segment = NIL THEN GO TO newseg;
    IF sp.page = p THEN EXIT;
    IF sp.lastused < minUseVal THEN
      BEGIN minUseVal←sp.lastused; minUseIndex←i END;
    REPEAT FINISHED =>
      BEGIN
      FOR i IN [0..maxsegments) DO
        CS[i].lastused ← CS[i].lastused - minUseVal;
        ENDLOOP;
      CurrentUseValue ← CurrentUseValue - minUseVal;
      sp ← @CS[minUseIndex];
      SegmentDefs.Unlock[sp.segment];
      SegmentDefs.DeleteFileSegment[sp.segment];
      sp.segment ← NIL;
      GO TO 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.lastused ← CurrentUseValue ← CurrentUseValue+1;
  CacheSwap.proc ← FlushCoreCache;
  RETURN[sp];
  END;

READ: PUBLIC PROCEDURE [a: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN OPEN AltoDefs, InlineDefs;
  fp: PageNumber = SELECT BITSHIFT[a,-LogPageSize] FROM
    IN [2..253] => BITSHIFT[a,-LogPageSize],
    1 => 254,
    0 => 255,
    ENDCASE => 0;
  IF fp=0 THEN RETURN [MEMORY[a]];
  IF PageMap[fp] = 0 THEN SIGNAL InvalidPage[fp];
  RETURN [(GetCS[fp].address + BITAND[a,PageSize-1])↑];
  END;

CopyRead: PUBLIC PROCEDURE [from, to: POINTER, nwords: CARDINAL] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..nwords) DO
    (to+i)↑ ← READ[from+i];
    ENDLOOP;
  RETURN
  END;
```

```
InvalidImageFile: PUBLIC SIGNAL = CODE;

-- initialization

InitializeImageCache: PUBLIC PROCEDURE [seg: SegmentDefs.FileSegmentHandle] =
  BEGIN OPEN AltoFileDefs, DiskDefs, SegmentDefs;
  image: POINTER TO ImageDefs.ImageHeader;
  map: POINTER TO normal ImageDefs.MapItem;
  i, mapIndex: CARDINAL ← 0;
  page, count, imagePage: CARDINAL;
  p: POINTER;
  ImageDAs: ARRAY [-1..256] OF AltoFileDefs.vDA;
  diskrequest: DiskRequest;
  SwapIn[seg];
  image ← FileSegmentAddress[seg];
  IF image.prefix.type = checkfile THEN SIGNAL InvalidImageFile;
  diskrequest ← DiskRequest [
    ca: SystemDefs.AllocatePages[1],
    fixedCA: TRUE,
    da: @ImageDAs[0],
    fp: @seg.file.fp,
    firstPage: 0,
    lastPage: 255,
    action: ReadD,
    lastAction: ReadD,
    signalCheckError: FALSE,
    option: update[BFSDefs.GetNextDA]];
  p ← LOOPHOLE[@ImageDAs[0]-1];
  p↑ ← fillinDA;
  InlineDefs.COPY[from: p, to: p+1, nwords: 257];
  ImageDAs[0] ← seg.file.fp.leaderDA;
  [] ← BFSDefs.ActOnPages[LOOPHOLE[@diskrequest]];
  SystemDefs.FreePages[diskrequest.ca];
  map ← LOOPHOLE[@image.map[0]];
  imagePage ← ImageDefs.FirstImageDataPage;
  FOR i IN [0..256) DO DAs[i] ← AltoFileDefs.eofDA; PageMap[i] ← 0; ENDLOOP;
  WHILE (map+mapIndex)↑ # ImageDefs.MapItem[0,0,normal[]] DO
    page ← (map+mapIndex)↑.page;
    count ← (map+mapIndex)↑.count;
    FOR i IN [0..count) DO
      DAs[page+i] ← ImageDAs[imagePage+i];
      PageMap[page+i] ← imagePage+i;
      ENDLOOP;
    imagePage ← imagePage+count;
    mapIndex ← mapIndex + 1;
    ENDLOOP;
  Unlock[seg];
  InitImageCache[seg.file];
  END;

END...
```