```
-- File: fakesegrun.mesa
-- Edited by: Sandman, April 19, 1978  3:53 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  InlineDefs: FROM "inlinedefs",
  Mopcodes: FROM "mopcodes",
  SegmentDefs: FROM "segmentdefs",
  WartDefs: FROM "wartdefs",
  FakeSegDefs: FROM "fakesegdefs";

DEFINITIONS FROM AltoDefs, SegmentDefs, WartDefs, FakeSegDefs;

FakeSegRun: PROGRAM IMPORTS SegmentDefs EXPORTS FakeSegDefs SHARES FakeSegDefs =
  BEGIN

  VMFileHandle: PUBLIC FileHandle ← NIL;

  FakeTooManySegments: PUBLIC SIGNAL = CODE;

  FakeVMNotFree: PUBLIC SIGNAL [base: PageNumber, pages: PageCount] = CODE;
  FakeInvalidSegmentSize: PUBLIC ERROR = CODE;
  FakeInsufficientVM: PUBLIC SIGNAL [needed:PageCount] = CODE;

  GetSegmentBootLink: PUBLIC PROCEDURE [s:FakeSegmentHandle] RETURNS [SegmentBootIndex] =
    BEGIN FakeValidateSeg[s]; RETURN[LOOPHOLE[s.BootLink, SegmentBootIndex]] END;

  SetSegmentBootLink: PUBLIC PROCEDURE [s:FakeSegmentHandle, i: BootIndex] =
    BEGIN FakeValidateSeg[s]; s.BootLink ← i END;

  FakeNewSegment: PUBLIC PROCEDURE [
    file:FileHandle, base:PageNumber, pages:PageCount, access:AccessOptions]
    RETURNS [s:FakeSegmentHandle] =
    BEGIN
    IF file=DefaultFile THEN file ← VMFileHandle;
    IF access=DefaultAccess THEN access ← GetFileAccess[file];
    WHILE FreeSegmentList=NIL DO
      SIGNAL FakeTooManySegments;
      ENDLOOP;
    s ← FreeSegmentList;
    FreeSegmentList ← s.Link;
      BEGIN ENABLE UNWIND =>
        BEGIN
        s.Link ← FreeSegmentList;
        FreeSegmentList ← s;
        END;
      IF file=VMFileHandle THEN
        BEGIN
        IF pages=DefaultPages OR pages<=0
          THEN ERROR FakeInvalidSegmentSize;
        IF base=DefaultBase THEN base ← AllocVM[pages,-1]
        ELSE
          BEGIN
          WHILE ~PagesFree[base,pages] DO
            SIGNAL FakeVMNotFree[base,pages];
            ENDLOOP;
          ReserveVM[base,pages]
          END;
        END
      ELSE
        BEGIN
        [] ← GetFileAccess[file]; -- instead of ValidateFile[file];
        IF base=DefaultBase THEN base ← 1;
        IF pages=DefaultPages THEN pages ← GetEndOfFile[file].page-base+1;
        END;
      END;
    s↑ ← FakeSegmentObject [
      file,file,other,access,FALSE,FALSE,base,pages,0,NIL,0,NIL,0,NIL,NIL];
    IF file # VMFileHandle THEN file.segcount ← file.segcount+1;
    IF file=VMFileHandle THEN
      BEGIN
      s.SwappedIn ← TRUE;
      s.VMaddress ← AddressFromPage[base];
      s.LockCount ← 1;
      END;
    RETURN
```

```
    END;

  FakeDeleteSegment: PUBLIC PROCEDURE [s:FakeSegmentHandle] =
    BEGIN
    f: FileHandle ← s.File;
    FakeValidateSeg[s];
    IF f=VMFileHandle THEN
      s.LockCount ← s.LockCount-1;
    FakeSwapOut[s];
    IF f # VMFileHandle AND (f.segcount ← f.segcount-1) = 0
      THEN ReleaseFile[f];
    s.Type ← free;
    s.Link ← FreeSegmentList;  FreeSegmentList ← s;
    RETURN
    END;


  -- Swapping Segments (but faking it)

  FakeSwapError: PUBLIC ERROR [s:FakeSegmentHandle] = CODE;

  FakeSwapIn: PUBLIC PROCEDURE [s:FakeSegmentHandle] =
    BEGIN
    vmpage: PageNumber;
    FakeValidateSeg[s];
    IF ~s.SwappedIn THEN
      BEGIN
      IF s.Pages=0 THEN ERROR FakeSwapError[s];
      vmpage ← AllocVM[s.Pages,1];
      s.VMaddress ← AddressFromPage[vmpage];
      s.SwappedIn ← TRUE;
      END;
    s.LockCount ← s.LockCount+1;
    RETURN
    END;

  FakeUnlock: PUBLIC PROCEDURE [s:FakeSegmentHandle] =
    BEGIN
    FakeValidateSeg[s];
    IF s.LockCount=0
      THEN ERROR FakeSwapError[s];
    s.LockCount ← s.LockCount-1;
    RETURN
    END;

  FakeSwapOut: PUBLIC PROCEDURE [s:FakeSegmentHandle] =
    BEGIN  OPEN InlineDefs;
    FakeValidateSeg[s];
    IF ~s.SwappedIn THEN RETURN;
    IF s.Pages=0 THEN ERROR FakeSwapError[s];
    IF s.LockCount > 0
      THEN ERROR FakeSwapError[s];
    ReleaseVM[PageFromAddress[s.VMaddress],s.Pages];
    s.SwappedIn ← FALSE;  s.VMaddress ← 0;
    RETURN
    END;


  -- VM Suboutines

  PageFree: INTEGER = 0;
  PageReserved: INTEGER = 1;
  PageState: TYPE = [PageFree..PageReserved];
  PageID: TYPE = WORD;
  PageMapHandle: TYPE = POINTER TO PageMap;
  PageMap: TYPE = ARRAY [0..AltoDefs.MaxVMPage/AltoDefs.wordlength] OF WORD;

  VMmap: PageMap ← [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

  MakePageID: PROCEDURE [PageNumber] RETURNS [PageID] =
    MACHINE CODE BEGIN Mopcodes.zLI4; Mopcodes.zSHIFT END;

  GetPageState: PROCEDURE [page:PageNumber] RETURNS [PageState] =
    BEGIN OPEN InlineDefs;
    ReadPageState: PROCEDURE [PageMapHandle, PageID] RETURNS [PageState] =
      MACHINE CODE BEGIN Mopcodes.zRFS END;
```

```
  RETURN[ReadPageState[@VMmap, MakePageID[page]]];
  END;

SetPageState: PROCEDURE [page:PageNumber, state:PageState] =
  BEGIN OPEN InlineDefs;
  WritePageState: PROCEDURE [PageState, PageMapHandle, PageID] =
    MACHINE CODE BEGIN Mopcodes.zWFS END;
  WritePageState[state, @VMmap, MakePageID[page]];
  RETURN;
  END;

ffvmp, lfvmp, minap, maxap: PageNumber;

AllocVM: PROCEDURE [pages:PageCount, direction:INTEGER] RETURNS [PageNumber] =
  BEGIN  n: INTEGER;
  pg, end, base: PageNumber;
  IF pages ~IN (0..MaxVMPage+1] THEN
    ERROR FakeInvalidSegmentSize;
  DO -- repeat if insufficient VM
    IF direction>0 THEN
      BEGIN  direction ← 1;
      -- eliminate any prefix of allocated pages and update ffvmp
      FOR ffvmp INCREASING IN [ffvmp..lfvmp] DO
        IF GetPageState[ffvmp]=PageFree THEN EXIT;
        ENDLOOP;
      pg ← ffvmp; end ← lfvmp;
      END
    ELSE
      BEGIN  direction ← -1;
      -- eliminate any suffix of allocated pages and update lfvmp
      FOR lfvmp DECREASING IN [ffvmp..lfvmp] DO
        IF GetPageState[lfvmp]=PageFree THEN EXIT;
        ENDLOOP;
      pg ← lfvmp; end ← ffvmp;
      END;
    n ← 0;  -- count of contiguous free pages
    FOR pg ← pg, pg+direction
    UNTIL (IF direction>0 THEN pg>end ELSE pg<end) DO
      IF GetPageState[pg]#PageFree
        THEN n ← 0  -- page in use; reset free count
        ELSE IF (n ← n+1) = pages
          THEN BEGIN
          base ← IF direction>0 THEN pg-n+1 ELSE pg;
          ReserveVM[base,pages];
          RETURN[base]
          END;
      ENDLOOP;
    SIGNAL FakeInsufficientVM[pages];
    ENDLOOP
  END;

ReleaseVM: PROCEDURE [base:PageNumber, pages:PageCount] =
  BEGIN
  ffvmp ← MIN [ffvmp,base];
  lfvmp ← MAX [lfvmp,base+pages-1];
  FOR base IN [base..base+pages) DO
    SetPageState[base,PageFree];
    ENDLOOP;
  RETURN
  END;

ReserveVM: PROCEDURE [base:PageNumber, pages:PageCount] =
  BEGIN
  FOR base IN [base..base+pages) DO
    SetPageState[base,PageReserved];
    ENDLOOP;
  RETURN
  END;

SetVMBounds: PROCEDURE [fp,lp:PageNumber] =
  BEGIN
  minap ← ffvmp ← fp;
  maxap ← lfvmp ← lp;
  RETURN
  END;
```

```
GetVMBounds: PUBLIC PROCEDURE RETURNS [VMBounds] =
  BEGIN
  FOR ffvmp INCREASING IN [ffvmp..lfvmp] DO
    IF GetPageState[ffvmp]=PageFree THEN EXIT;
    ENDLOOP;
  FOR lfvmp DECREASING IN [ffvmp..lfvmp] DO
    IF GetPageState[lfvmp]=PageFree THEN EXIT;
    ENDLOOP;
  IF ~PagesFree[ffvmp,lfvmp-ffvmp+1] THEN
    SIGNAL FakeVMNotFree[ffvmp,lfvmp-ffvmp+1];
  RETURN[[ffvmp, lfvmp]];
  END;

PagesFree: PROCEDURE [base:PageNumber, pages:PageCount] RETURNS [BOOLEAN] =
  BEGIN
  FOR base IN [base..base+pages) DO
    IF GetPageState[base]#PageFree THEN RETURN[FALSE];
    ENDLOOP;
  RETURN[TRUE]
  END;

-- kludges for peering into machine addresses
PAGEDISP: TYPE = MACHINE DEPENDENT RECORD [
  page: [0..MaxVMPage],
  disp: [0..PageSize)];

PageFromAddress: PROCEDURE [a:UNSPECIFIED] RETURNS [PageNumber] =
  BEGIN
  word: PAGEDISP ← a;
  RETURN[word.page]
  END;

AddressFromPage: PROCEDURE [p:PageNumber] RETURNS [UNSPECIFIED] =
  BEGIN
  RETURN[PAGEDISP[p,0]]
  END;


-- Segment Sub-Routines

NumberOfSegments: INTEGER = 200;
FreeSegmentList: FakeSegmentHandle;
SegmentIndex: TYPE = [0..NumberOfSegments);
SegArray: ARRAY SegmentIndex OF FakeSegmentObject;

FakeInvalidSegment: PUBLIC ERROR [POINTER] = CODE;

FakeValidateSeg: PUBLIC PROCEDURE [s:FakeSegmentHandle] =
  BEGIN OPEN InlineDefs;
  i, j: INTEGER;
  [i, j] ← DIVMOD [
    s-@SegArray[FIRST[SegmentIndex]],SIZE[FakeSegmentObject]];
  IF i ~IN SegmentIndex OR j#0 OR s.Type=free
    THEN ERROR FakeInvalidSegment[s];
  RETURN
  END;

FakeInitSegMachinery: PUBLIC PROCEDURE [firstpage,lastpage:PageNumber] =
  BEGIN  i: SegmentIndex;
  SetVMBounds[firstpage,lastpage];
  FreeSegmentList ← @SegArray[FIRST[SegmentIndex]];
  FOR i IN SegmentIndex DO
    SegArray[i].Type ← free;
    SegArray[i].Link ← @SegArray[i+1];
    ENDLOOP;
  SegArray[LAST[SegmentIndex]].Link ← NIL;
  RETURN
  END;

FakeEnumerateSegments: PUBLIC PROCEDURE [
  proc:PROCEDURE[FakeSegmentHandle]RETURNS[BOOLEAN] ]
  RETURNS [FakeSegmentHandle] =
  BEGIN
  i: SegmentIndex;
  s: FakeSegmentHandle;
  FOR i IN SegmentIndex DO
```

```
      IF (s ← @SegArray[i]).Type # free THEN
        IF proc[s] THEN RETURN [s];
      ENDLOOP;
    RETURN[NIL]
    END;

END...
```