OS-X/Zylix
Operating system for CPCNG

Technical documentation and source code project.

Note : A NGBASIC interpreter will be include in the shell of OS-X Zylix.
Source code of the OS in TML2 Language.

osx20.bin	OS-X 2.0 kernel, file servers and disk image file.
support.o	TML2 runtime support library in object format.
zemu32.exe	Z180 emulator for Windows 95/98/NT/2000.
sys.tml	OS-X 2.0 system call interface (API).
sys_rpc.tml	OS-X 2.0 system definitions, RPC messages etc.
file_rpc.tml	OS-X 2.0 file set/get attributes library.
hex_str.tml	Hex dump library.
io.tml	Formatted text input/output library.
memory.tml	Absolute memory and I/O access library.
output.tml	Formatted text output library.
stdlib.tml	OS-X 2.0 kernel services standard library.
cat.tml	Source code for file print/concatenation program.
du.tml	Source code for disk usage program.
hd.tml	Source code for hex dump program.
*.o *.sym	Pre-compiled objects and symbol listings for source above.

After booting a prompt like this will appear:

root_sh>

Try the help command for a quick summary of commands.

A file tree similar to UNIX exist with this content:

/ root directory

/dev a directory where all device special files are stored

/bin a read-only directory with some useful programs

/usr another directory with some test files

/tmp a directory for temporary storage

/host a read-only directory which gives access to PC-files

Try these commands:

cd /host

ls -l

And you should see a listing of the files on your CPCNG.
There are some useful programs under /bin like:

zsh a command shell with many built in commands and command line history and editing (emacs-style).

Note: zshell has a command line history and supports command line editing

with emacs key strokes. CTRL-P for previous command, CTRL-N for next

command, CTRL-B for character backwards, CTRL-F for char. forward etc.

```
cat             a small program that reads a text file given as
                parameter and writes the text to stdout.
cp              a file copy program.
du              display disk usage by reading directories
recursively
                from a given starting point (or current dir).
hd              hex dump program.
emacs           a very small emacs look-a-like text editor.
ovl_main    a test program using TML2 automatic overlays.
zfsd            the host file system access server,
                started at boot.
forth           a small Forth interpreter.
To start a program, just type its file name followed by any
parameters
and hit carriage return. If the program is not in the search
path or
the current directory you must type the full path, like this:
/bin/hd myfile
```

CAT.TML

```
{*
 *  File concatenation program for OS-X
 *  Version 94-07-15
 *}

UNIT cat;

IMPLEMENTATION

 uses
   sys,
   sys_rpc;

 var
   i,n,f,x : int;
   ignore  : int;
   line    : string;


  PROCEDURE putchar(ch: char);
  BEGIN
    ignore := write(1,@ch,1);
  END;
```

```
PROCEDURE puts(s: string);
BEGIN
  ignore := write(1,@s[1],length(s));
END;



PROCEDURE do_cat();
BEGIN
  if argc() > 1 then
     for i := 1 while i < argc() do inc(i) loop
        f := open(argv(i)^,o_read);
        if f <> -1 then
           repeat
              n := read(f,@line,sizeof(line));
              x := write(1,@line,n);
           until n = 0;
           close(f);
        else
           puts("cat: ");
           puts(argv(i)^);
           puts(": couldn't open.");
           putchar(#13);
           putchar(#10);
           exit(-1);
        end;
```

```
       end loop;
    else
      puts("usage: cat <filenames>");
      putchar(#13);
      putchar(#10);
    end;
    exit(0);
  END do_cat;



BEGIN
  do_cat();
END cat;
```

```
DU.TML
{*
 *  du.tml
 *  $Id: du.tml 1.1 1999/05/27 09:39:02 frago Exp frago $
 *
 *  Disk usage program for OS-X.
 *
 *  Revision history:
 *    990526 Francis Gormarker, initial issue.
 *}


UNIT du;


IMPLEMENTATION


  uses
    sys,
    sys_rpc,
    stdlib,
    memory,
    output;


  var
    debug      : boolean;
    fd,t0,t1   : int;
    attr       : fattr_t;
```

```
  entry      : dir_entry;
  pbuf       : string;


PROCEDURE round_up(n: long): long;
 var
   diff : long;
BEGIN
  diff := n and long(255);
  if diff <> long(0) then
     return n + long(256) - diff;
  end;
  return n;
END;


PROCEDURE recurse(path: string): long;
 var
   fd,len,n,nsize,pos : int;
   count              : long;
BEGIN
  count := long(0);
  if stat(path,@attr) = 0 then
     count := round_up(attr.size);
     if (attr.mode and S_IFMT) <> S_IFDIR then
        if debug then
```

```
            putln(int(count / long(1024))," kB ",path);
         end;
      else
         len := length(pbuf);
         fd := open(path,o_read);
         n := read(fd,@entry,sizeof(entry));
         while n <> 0 loop
            nsize := length(entry.name);
            if (nsize <> 0) and (string(entry.name) <> ".")
and (string(entry.name) <> "..") then
               if pbuf[len] <> '/' then
                  pbuf[len+1] := '/';
                  pos := len + 2;
               else
                  pos := len + 1;
               end;
               copy(@entry.name[1],@pbuf[pos],nsize);
               length(pbuf) := (pos - 1) + nsize;
               count := count + recurse(pbuf);
            end;
            n := read(fd,@entry,sizeof(entry));
         end loop;
         close(fd);
         length(pbuf) := len;
         putln(int(count / long(1024))," kB ",path);
      end;
```

```
  else
     putln("du: ",path,": stat failed.");
  end;
  return count;
END;



PROCEDURE do_du(path: string);
 var
   count : long;
BEGIN
  pbuf := path;
  if length(pbuf) = 0 then
     pbuf := ".";
  end;


  {
  if debug then
     putln("du: reading ",pbuf);
  end;
  }
  t0 := sys_time();
  count := recurse(pbuf);
  t1 := sys_time();
```

```
    {
    if debug then
       putln("du: ",int(count / long(1024))," kB took ",t1-
t0," ms.");
    end;
    }
    exit(0);
  END;



BEGIN
  if argv(1)^ = "-debug" then
     debug := true;
     do_du(argv(2)^);
  else
     debug := false;
     do_du(argv(1)^);
  end;
END du;
```

HD.TML

```
{*
 *  hd.tml
 *  $Id: hd.tml 1.2 1999/12/23 12:58:23 frago Exp frago $
 *
 *  Hex dump program for OS-X
 *
 *  Revision history:
 *    950824 Francis Gormarker, last OS-X version.
 *
 *    991205 FrGo, first port to OS-X 2.0
 *           BUGS: ASCII dump is not displayed for last line
if data size
 *           is not an even multiple of 16 bytes.
 *
 *    991222 FrGo, now using 256 byte buffer. Bugfix: now
using hex_str unit
 *           for correct hex display. Added address display.
 *}


UNIT hd;




IMPLEMENTATION


 uses
   sys,
```

```
  sys_rpc,
  memory,
  hex_str,
  output;

var
  ignore   : int;
  buf      : array[256] of char;


 PROCEDURE min(x,y: int): int;
 BEGIN
   if x < y then
      return x;
   else
      return y;
   end;
 END;


 FUNCTION val(s: string): int;
  var
    i,sum : int;
 BEGIN
    sum := 0;
```

```
   i := 1;
   while i <= length(s) loop
      sum := sum * 10 + int(s[i] - '0');
      inc(i);
   end loop;
   return sum;
END val;

{
const
  hextab : string = ('0','1','2','3','4','5','6','7','8',
                     '9','A','B','C','D','E','F');


PROCEDURE display_data(data: pointer; size: int);
 var
   i,n : int;
   ch  : char;
   p   : ^string;
   s   : array[17] of char;
BEGIN
  p := data;
  for i := 0 while i < size do inc(i) loop
      if (i mod 16) = 0 then
         {
         n := i;
```

```
   putchar(hextab[n / 4096]);
   n := n mod 4096;
   putchar(hextab[n / 256]);
   n := n mod 256;
   putchar(hextab[n / 16]);
   n := n mod 16;
   putchar(hextab[n]);
   put("  ");
   }
   fill(@s,sizeof(s),#0);
end;
ch := p^[i];
n := int(ch);
if (n < 32) or (n > 127) then
   ch := '.';
end;
s[i mod 16 + 1] := ch;
put(hextab[n / 16]);
n := n mod 16;
put(hextab[n]);
put(' ');
if (i mod 16) = 15 then
   length(s) := 16;
   putln(string(s));
end;
```

```
  end loop;
  if (i mod 16) <> 0 then
     putln();
  end;
END;
}



PROCEDURE hexdump(name: string; start,max: int);
 var
   fd,n,offset,pos,i : int;
BEGIN
  fd := open(name,o_read);
  if fd = -1 then
     putln("hd: ",name,": couldn't open.");
  else
     putln("hd: dump of """,name,""" at ",start);
     lseek(fd,long(start),SEEK_SET);
     n := read(fd,@buf,min(256,max));
     offset := 0;
     {
     pos := start;
     if (start and 15) <> 0 then
        putln();
        put(start,"  ");
     end;
```

```
        }
        while (n <> 0) and (offset < max) loop
           {
           if (pos and 15) = 0 then
              putln();
              put(pos,"  ");
           end;
           }
           i := 0;
           while i < n loop
             put(itox(start+offset+i)^,"  ");
             hex_dump(@buf[i],min(n-i,16));
             i := i + 16;
           end loop;
           n := read(fd,@buf,min(256,max-offset));
           offset := offset + n;
        end loop;
        close(fd);
     end;
  END;



BEGIN
  if argc() = 2 then
     hexdump(argv(1)^,0,65535);
```

```
  elsif argc() = 3 then
     hexdump(argv(1)^,val(argv(2)^),65535);
  elsif argc() = 4 then
     hexdump(argv(1)^,val(argv(2)^),val(argv(3)^));
  else
     putln("Usage: hd file [ start ] [ maxlen ]");
  end;
END hd;
```

HEX_STR.TML

```
{*
 *  hex_str.tml
 *  $Id: hex_str.tml 1.2 1999/04/11 23:31:57 frago Exp frago $
 *
 *  Hex string conversion unit.
 *
 *  Revision history:
 *    980911 Francis Gormarker, initial issue.
 *
 *    990411 FrGo, improved performance of hex_dump(). Bugfix: the last
 *           data bytes were not dumped as ASCII if data size not an even
 *           multiple of 16.
 *}

UNIT hex_str;

INTERFACE

  PROCEDURE itox(n: int): ^string;
  PROCEDURE hex_dump(data: pointer; size: int);

IMPLEMENTATION

 uses
```

```
memory,
system;



const
  hextab : string = ('0','1','2','3','4','5','6','7','8',
                     '9','A','B','C','D','E','F');

var
  hexbuf : array[8] of char;



PROCEDURE itox(n: int): ^string;
BEGIN
  hexbuf[1] := hextab[n / 4096];
  n := n mod 4096;
  hexbuf[2] := hextab[n / 256];
  n := n mod 256;
  hexbuf[3] := hextab[n / 16];
  n := n mod 16;
  hexbuf[4] := hextab[n];
  length(hexbuf) := 4;
  return (@hexbuf);
END;
```

```
PROCEDURE hex_dump(data: pointer; size: int);
 var
   i,j,k,n : int;
   ch      : char;
   p       : ^string;
   s       : array[80] of char;
BEGIN
  p := data;
  for i := 0 while i < size do inc(i) loop
      j := i and 15;
      if j = 0 then
         {
         n := i;
         putchar(hextab[n / 4096]);
         n := n mod 4096;
         putchar(hextab[n / 256]);
         n := n mod 256;
         putchar(hextab[n / 16]);
         n := n mod 16;
         putchar(hextab[n]);
         put("  ");
         }
         fill(@s,sizeof(s),' ');
         length(s) := 64;
         k := 1;
```

```
        end;
        ch := p^[i];
        n := int(ch);
        if (n < 32) or (n > 127) then
           ch := '.';
        end;
        s[j + 49] := ch;
        s[k] := hextab[shr(n,4)];
        s[k + 1] := hextab[n and 15];
        k := k + 3;
        if j = 15 then
           putln(string(s));
        end;
    end loop;
    if (i mod 16) <> 0 then
       putln(string(s));
    end;
  END;


BEGIN
END hex_str;
```

```
INPUT.TML
{*
 *  input.tml
 *  $Id: input.tml 1.2 2001/06/07 09:59:02 frago Exp frago $
 *
 *  Terminal input unit with line editor and history list.
 *
 *  Revision history:
 *    940829 Francis Gormarker, original OS-X version.
 *
 *    990411 FrGo, modified for OS-X 2.0
 *
 *    010607 FrGo, added support for VT100 cursor keys.
 *}

UNIT input;
INTERFACE

  PROCEDURE getLine(var s: string; max: int);
  PROCEDURE init_input();

IMPLEMENTATION

  uses
    sys,
    {sys_rpc,}
```

```
  {output,}
  memory;

const
  max_history = 16;

type
  hstring = array[64] of char;

const
  ctrlSP = #0;
  ctrlA = #1;
  ctrlB = #2;
  ctrlC = #3;
  ctrlD = #4;
  ctrlE = #5;
  ctrlF = #6;
  ctrlG = #7;
  ctrlH = #8;
  ctrlI = #9;
  ctrlJ = #10;
  ctrlK = #11;
  ctrlL = #12;
  ctrlM = #13;
  ctrlN = #14;
```

```
  ctrlO = #15;
  ctrlP = #16;
  ctrlQ = #17;
  ctrlR = #18;
  ctrlS = #19;
  ctrlT = #20;
  ctrlU = #21;
  ctrlV = #22;
  ctrlW = #23;
  ctrlX = #24;
  ctrlY = #25;
  ctrlZ = #26;
  esc   = #27;
  lf    = #10;
  cr    = #13;
  bs    = #8;

var
  ignore  : int;
  hrd,hwr : int;
  hindex  : int;
  {line    : string;}
  hist    : array[max_history] of hstring;


PROCEDURE putchar(ch: char);
```

```
BEGIN
  write(1,@ch,1);
END;



PROCEDURE getchar(): int;
 var
   ch : char;
BEGIN
  if read(0,@ch,1) = 1 then
     return int(ch);
  else
     return -1;
  end;
END;



{
PROCEDURE clearln(max: int);
 var
   i : int;
BEGIN
  putchar(#13);
  for i := 0 while i < max do inc(i) loop
      putchar(' ');
```

```
    end loop;
    putchar(#13);
  END;
}

  PROCEDURE bwd(n: int);
   var
     i : int;
  BEGIN
    for i := 0 while i < n do inc(i) loop
        putchar(#8);
    end loop;
  END;



  PROCEDURE cleoln(max: int);
   var
     i : int;
  BEGIN
    for i := 0 while i < max do inc(i) loop
        putchar(' ');
    end loop;
    bwd(max);
  END;
```

```
PROCEDURE update(s: string; pos,size: int);
 var
   i : int;
BEGIN
  ignore := write(1,@s[pos],1+size-pos);
  putchar(' ');
  bwd(1+size-pos);
END;



PROCEDURE add_history(s: string);
BEGIN
  hist[hwr] := s;
  hwr := (hwr + 1) mod max_history;
  if hwr = hrd then
     hrd := (hrd + 1) mod max_history;
  end;
END;



PROCEDURE current_history();
BEGIN
  hindex := hwr;
END;
```

```
PROCEDURE prev_history();
BEGIN
  if hindex <> hrd then
     hindex := (hindex - 1) mod max_history;
  end;
END;



PROCEDURE next_history();
BEGIN
  if hindex <> hwr then
     hindex := (hindex + 1) mod max_history;
  end;
END;



PROCEDURE get_history(): ^string;
BEGIN
  if hindex <> hwr then
     return @hist[hindex];
  else
     return pointer("");
  end;
END;
```

```
PROCEDURE getLine(var s: string; max: int);
 var
   size : int;
   i    : int;
   ch   : int;
BEGIN
  current_history();
  size := length(s);
  i := size+1;
  ch := 0;
  if max > 63 then
     max := 63;
  end;
  ignore := write(1,@s[1],size);
  loop
    ch := getchar();
    if ch = -1 then
       ch := 13;
    end;
    if char(ch) = esc then
       ch := getchar();
       if char(ch) = '[' then
          ch := getchar();
          case char(ch) of
```

```
         'A' : ch := int(ctrlP);  { Cursor up }
         'B' : ch := int(ctrlN);  { Cursor down }
         'C' : ch := int(ctrlF);  { Cursor right }
         'D' : ch := int(ctrlB);  { Cursor left }
         'H' : ch := int(ctrlA);  { Home }
      end case;
   end;
end;
if char(ch) >= ' ' then
   if size+1 < max then
      copy(@s[i],@s[i+1],1+size-i);
      s[i] := char(ch);
      inc(size);
      update(s,i,size);
      inc(i);
   end;
else
  case char(ch) of
     #8 : begin
        { Backspace }
        if (i > 1) then
           copy(@s[i],@s[i-1],1+size-i);
           dec(i);
           dec(size);
           putchar(#8);
           if i = size+1 then
```

```
            putchar(' ');
         else
            update(s,i,size);
         end;
         putchar(#8);
      end;
   end;
   ctrlB : begin
      { Move backward }
      if (i > 1) then
         dec(i);
         putchar(#8);
      end;
   end;
   ctrlD : begin
      { Delete current char }
      if (size+1 > i) then
         copy(@s[i+1],@s[i],1+size-i);
         dec(size);
         update(s,i,size);
         putchar(#8);
      end;
   end;
   ctrlF : begin
      { Move forward }
```

```
   if (i < size+1) then
      putchar(s[i]);
      inc(i);
   end;
end;
ctrlA : begin
   { Move to first }
   bwd(i-1);
   i := 1;
end;
ctrlE : begin
   { Move to last }
   ignore := write(1,@s[i],1+size-i);
   i := size+1;
end;
ctrlP : begin
   bwd(i-1);
   cleoln(size);
   prev_history();
   s := get_history()^;
   {clearln(max);}
   size := length(s);
   i := size+1;
   ignore := write(1,@s[1],size);
end;
ctrlN : begin
```

```
      bwd(i-1);
      cleoln(size);
      next_history();
      s := get_history()^;
      {clearln(max);}
      size := length(s);
      i := size+1;
      ignore := write(1,@s[1],size);
   end;
   ctrlL : begin
      { Refresh line }
      bwd(i-1);
      cleoln(max);
      ignore := write(1,@s[1],size);
      bwd(1+size-i);
   end;
   #13 : begin
      { Carriage Return }
      length(s) := size;
      if size <> 0 then
         add_history(s);
      end;
      putchar(#13);
      putchar(#10);
      return;
```

```
            end;
          end case;
        end if;
      end loop;
    END getLine;



    PROCEDURE init_input();
    BEGIN
      hrd := 0;
      hwr := 0;
      hindex := 0;
    END;



BEGIN
    {
    putln("Input test.");
    init_input();
    loop
      put("input>");
      line := "";
      getLine(line,70);
      putln("You typed '",line,"'");
    end loop;
    }
```

```
END input;
```

IO.TML

```
{*
 *  io.tml
 *  $Id: io.tml 1.2 1999/06/15 19:40:04 frago Exp frago $
 *
 *  Small input/output unit for OS-X 2.0
 *
 *  Revision history:
 *    940714 Francis Gormarker, last OX-X 1.0 issue.
 *
 *    990316 FrGo, modified for OS-X 2.0
 *
 *    990615 FrGo, modified gets() to return EOF status.
 *}

UNIT io;

INTERFACE

  FUNCTION decstr(n: int; var res: string);
  {PROCEDURE fgets(f: pointer; var s: string);}

  PROCEDURE fputs(fd: int; s: string);
  PROCEDURE gets(var s: string): int;
  PROCEDURE put(GENERIC);
  PROCEDURE putln(GENERIC);
```

```
IMPLEMENTATION

  uses
    sys;

  var
    ignore : int;



  PROCEDURE _putchar(ch: char);
  BEGIN
    ignore := write(1,@ch,1);
  END;



  PROCEDURE _getchar(): int;
   var
     ch : char;
  BEGIN
    if read(0,@ch,1) = 1 then
       return int(ch);
    else
       return -1;
    end;
```

```
END;


{ Convert n to right adjusted decimal string,
  skip leading zeros. }

FUNCTION decstr(n: int; var res: string);
 var
  b,i,x : int;
  go    : boolean;
BEGIN
  go := false;
  i := 1;
  b := 10000;
  while b <> 0 loop
    x := n / b;
    if x > 0 then
       go := true;
    end;
    if go or (b = 1) then
       res[i] := '0' + char(x);
       inc(i);
    else
       res[i] := ' ';
       inc(i);
    end;
```

```
      n := n mod b;
      b := b / 10;
    end;
    length(res) := i-1;
  END decstr;



{
  PROCEDURE fgets(f: pointer; var s: string);
   var
     i  : int;
     ch : char;
  BEGIN
    i := 1;
    repeat
      ch := getc(f);
      if (i < sizeof(string)) and (ch >= ' ') then
         s[i] := ch;
         inc(i);
      end;
    until eof(f) or (ch = #13);
    length(s) := i-1;
  END;
}
```

```
PROCEDURE fputs(fd: int; s: string);
BEGIN
  ignore := write(fd,@s[1],length(s));
END;



{ Return zero if ok or -1 if EOF }
PROCEDURE gets(var s: string): int;
 var
   i  : int;
   ch : int;
BEGIN
  i := 1;
  ch := _getchar();
  while (char(ch) <> #13) and (ch <> -1) loop
    if char(ch) >= ' ' then
       if i < 80 then
          _putchar(char(ch));
          s[i] := char(ch);
          i := i + 1;
       end;
    else
      if (char(ch) = #8) then
         if (i > 1) then
            _putchar(#8);
            _putchar(' ');
```

```
            _putchar(#8);
            i := i - 1;
         end;
      end;
    end;
    ch := _getchar();
  end;
  _putchar(#13);
  _putchar(#10);
  length(s) := i-1;
  if ch = -1 then
     return -1;
  end;
  return 0;
END gets;



PROCEDURE do_put(value,typeCode: int);
 var
   s : array[20] of char;
BEGIN
   case typeCode of
     1 : _putchar(char(value));
     2 : begin
           decstr(value,s);
```

```
            fputs(1,s);
          end;
      3 : if boolean(value) then
             fputs(1,"true");
          else
             fputs(1,"false");
          end;
      4 : begin
            decstr(value,s);
            fputs(1,s);
          end;
      5 : fputs(1,string(value));
    end;
END;



PROCEDURE put(GENERIC);
 var
   i : int;
BEGIN
  for i := 1 while i <= length(GENERIC) do inc(i) loop
      do_put(GENERIC[i].value,GENERIC[i].typeCode);
  end;
END put;
```

```
  PROCEDURE putln(GENERIC);
   var
     i : int;
  BEGIN
    for i := 1 while i <= length(GENERIC) do inc(i) loop
        do_put(GENERIC[i].value,GENERIC[i].typeCode);
    end;
    _putchar(#13);
    _putchar(#10);
  END putln;



BEGIN
END io;
```

IOLIB.TML

```
{*
 *  iolib.tml
 *  $Id: iolib.tml 1.1 1999/12/12 23:34:28 frago Exp frago $
 *
 *  I/O control library for OS-X 2.0
 *
 *  Revision history:
 *    991212 Francis Gormarker, initial issue.
 *
 *}
UNIT iolib;
INTERFACE

  PROCEDURE ioctl(fd,cmd: int; args: pointer; size: int): int;

IMPLEMENTATION

  uses
    sys,
    sys_rpc;


  PROCEDURE ioctl(fd,cmd: int; args: pointer; size: int): int;
   var
     fp     : gfd_ptr;
```

```
    server : int;
    msg    : file_ioctl_msg2;
  BEGIN
    fp := thread_get_gfd(fd);
    if fp = nil then
       return -1;
    end;
    server := ufid_lookup(@fp^.ufid);
    msg.cmd := cmd;
    msg.ufid := fp^.ufid;
    msg.uid := get_uid();
    return rpc_call(mod_filed,
                    file_ioctl,
                    @msg,
                    args,
                    server,
                    sizeof(msg),
                    size,
                    size,
                    10000);
  END;


BEGIN
END iolib;
```

```
MEMORY.TML
{ Unit for absolute memory manipulation ver 1995-05-10
  Modified for Z80180 16 bit I/O addresses }

UNIT memory;

INTERFACE

  FUNCTION out(port: int; data: char);
  FUNCTION in(port: int): char;
  FUNCTION call(funptr: pointer);
  FUNCTION set(address: int; data: int);
  FUNCTION copy(source,dest: pointer; size: int);
  FUNCTION fill(dest: pointer; size: int; data: char);

IMPLEMENTATION


  FUNCTION out(port: int; data: char);
  BEGIN
    inline($FD,$4E,4,$FD,$6E,6,$06,$00,$ED,$69);
  END;


  FUNCTION in(port: int): char;
  BEGIN
```

```
  inline($FD,$4E,4,$06,$00,$ED,$68);
END;



FUNCTION call(funptr: pointer);
 type
   fn = function();
BEGIN
  fn(funptr)();
END call;



{ Store data at absolute memory address }

FUNCTION set(address: int; data: int);
 type
   ptr = ^int;
BEGIN
  ptr(address)^ := data;
  {inline($FD,$E1,$DD,$E1,$D1,$E1,$73,$23,$72,$DD,$E9);}
END set;



{FUNCTION set(address: word; data: byte);
BEGIN
```

```
    inline($DD,$E1,$D1,$E1,$73,$DD,$E9);
  END set;}


  { Copy memory contents from source address to destination of
specified length
    in bytes. Handles overlapping areas. This one is FAST }

  FUNCTION copy(source,dest: pointer; size: int);
  BEGIN
    if size > 0 then
       if int(source) > int(dest) then
          inline($FD,$6E,4,$FD,$66,5);
          inline($FD,$5E,6,$FD,$56,7);
          inline($FD,$4E,8,$FD,$46,9);
          inline($ED,$B0);
       else
          source := pointer(int(source) + size - 1);
          dest := pointer(int(dest) + size - 1);
          inline($FD,$6E,4,$FD,$66,5);
          inline($FD,$5E,6,$FD,$56,7);
          inline($FD,$4E,8,$FD,$46,9);
          inline($ED,$B8);
       end if;
    end if;
  END copy;
```

```
  { Fill with 'data' byte from 'dest' address and 'size' bytes
forward. }


  FUNCTION fill(dest: pointer; size: int; data: char);

  BEGIN

    if size > 0 then

       inline($FD,$6E,4,$FD,$66,5);

       inline($FD,$5E,6,$FD,$56,7);

       inline($FD,$4E,8,$FD,$46,9);

       inline($71,$23,$1B,$7B,$B2,$20,$F9);

    end if;

  END fill;




BEGIN

END memory;
```

```
OUTPUT.TML
{*
 *  output.tml
 *  $Id: output.tml 1.1 1999/05/03 14:21:27 frago Exp frago $
 *
 *  Small output unit for OS-X 2.0
 *
 *  Revision history:
 *    990502 Francis Gormarker, initial OS-X 2.0 issue.
 *}

UNIT output;

INTERFACE

  FUNCTION decstr(n: int; var res: string);
  PROCEDURE fputs(fd: int; s: string);
  PROCEDURE put(GENERIC);
  PROCEDURE putln(GENERIC);

IMPLEMENTATION

  uses
    sys;

  var
```

```
  ignore : int;



PROCEDURE _putchar(ch: char);
BEGIN
  ignore := write(1,@ch,1);
END;



{ Convert n to right adjusted decimal string,
  skip leading zeros. }

FUNCTION decstr(n: int; var res: string);
 var
  b,i,x : int;
  go    : boolean;
BEGIN
  go := false;
  i := 1;
  b := 10000;
  while b <> 0 loop
    x := n / b;
    if x > 0 then
       go := true;
    end;
```

```
    if go or (b = 1) then
       res[i] := '0' + char(x);
       inc(i);
    else
       res[i] := ' ';
       inc(i);
    end;
    n := n mod b;
    b := b / 10;
  end;
  length(res) := i-1;
END decstr;



PROCEDURE fputs(fd: int; s: string);
BEGIN
  ignore := write(fd,@s[1],length(s));
END;



PROCEDURE do_put(value,typeCode: int);
 var
   s : array[20] of char;
BEGIN
   case typeCode of
     1 : _putchar(char(value));
```

```
      2 : begin
            decstr(value,s);
            fputs(1,s);
          end;
      3 : if boolean(value) then
             fputs(1,"true");
          else
             fputs(1,"false");
          end;
      4 : begin
            decstr(value,s);
            fputs(1,s);
          end;
      5 : fputs(1,string(value));
    end;
END;



PROCEDURE put(GENERIC);
 var
    i : int;
BEGIN
  for i := 1 while i <= length(GENERIC) do inc(i) loop
       do_put(GENERIC[i].value,GENERIC[i].typeCode);
  end;
```

```
  END put;



  PROCEDURE putln(GENERIC);
   var
      i : int;
  BEGIN
    for i := 1 while i <= length(GENERIC) do inc(i) loop
        do_put(GENERIC[i].value,GENERIC[i].typeCode);
    end;
    _putchar(#13);
    _putchar(#10);
  END putln;



BEGIN
END output;
```

```
STATLIB.TML
{*
 *  statlib.tml
 *  $Id: statlib.tml 1.1 1999/11/30 12:05:00 frago Exp frago $
 *
 *  File status library for OS-X 2.0
 *
 *  Revision history:
 *    991126 Francis Gormarker, initial issue.
 *}

UNIT statlib;
INTERFACE

  PROCEDURE stat(path: string; attr: pointer): int;

IMPLEMENTATION

  uses
    sys,
    sys_rpc,
    file_rpc;



  PROCEDURE stat(path: string; attr: pointer): int;
   var
```

```
   n       : int;
   info    : ^lookup_info;
   ufid    : ufid_t;
   msg     : kproc_message;
BEGIN
   {putln("stat(",path,")");}
   msg.data := path;
   n := length(msg.data) + 1;
   if n < sizeof(lookup_info) then
      n := sizeof(lookup_info);
   end;
   msg.cthread := thread_self();

   if rpc_call(mod_rootd,
               kproc_path_lookup,
               @msg,
               nil,
               kernel_port,
               sizeof(kproc_message2) + n,
               0,
               0,
               mq_infinite) = 0 then
      info := @msg.data;
      ufid := info^.ufid;
      return get_attr(@ufid,attr);
```

```
      end;
      return -1;
  END;



BEGIN
END statlib;
```

```
STDIO.TML

{*
 *  stdio.tml
 *  $Id: stdio.tml 1.3 2000/01/21 17:05:40 frago Exp frago $
 *
 *  POSIX standard buffered I/O library for OS-X 2.0
 *
 *  Revision history:
 *    990524 Francis Gormarker, initial issue.
 *
 *    991103 FrGo, added fseek, ftell, fread, fwrite
procedures.
 *
 *    000118 FrGo, bugfix: fwrite and fputc did not work
properly if
 *           used after fdopen call.
 *}


UNIT stdio;

INTERFACE


  PROCEDURE fdopen(fd: int): pointer;

  PROCEDURE fopen(path: string; mode: string): pointer;

  PROCEDURE fflush(fp: pointer): int;

  PROCEDURE fclose(fp: pointer): int;

  PROCEDURE fputc(c: int; fp: pointer): int;

  PROCEDURE fgetc(fp: pointer): int;
```

```
  PROCEDURE fseek(fp: pointer; offset: long; whence: int):
int;

  PROCEDURE ftell(fp: pointer): long;

  PROCEDURE fread(buf: pointer; size,items: int; fp: pointer):
int;

  PROCEDURE fwrite(buf: pointer; size,items: int; fp:
pointer): int;



IMPLEMENTATION



  uses

    sys,

    sys_rpc,

    output,

    memory;




  const

    max_buffer    = 256;



  type

    int_ptr  = ^int;

    data_ptr = ^array[256] of char;



    file    = record

                lpos  : long;

                pos   : int;
```

```
              size  : int;
              fd    : int;
              dirty : boolean;
              buf   : array[max_buffer] of char;
            end;

  file_ptr = ^file;



PROCEDURE fdopen(fd: int): pointer;
 var
   fp : file_ptr;
BEGIN
  fp := mem_alloc(sizeof(file));
  if fp <> nil then
     fill(fp,sizeof(file),#0);
     fp^.fd := fd;
     fp^.dirty := false;
  end;
  return fp;
END;



PROCEDURE fopen(path: string; mode: string): pointer;
```

```
 var
   flags : int;
   fd    : int;
   fp    : file_ptr;
BEGIN
  fp := mem_alloc(sizeof(file));
  if fp <> nil then
     fill(fp,sizeof(file),#0);
     flags := 0;
     if mode[1] = 'r' then
        flags := flags or O_READ;
     end;
     if mode[1] = 'w' then
        flags := flags or O_WRITE;
     end;
     if (mode = "r+") or (mode = "rb+") then
        flags := O_RDWR;
     end;
     fd := open(path,flags);
     if fd <> -1 then
        fp^.fd := fd;
        fp^.dirty := false;
     else
        mem_release(fp);
        fp := nil;
     end;
```

```
  end;
  return fp;
END;


PROCEDURE fflush(fp: pointer): int;
 var
   n,wsize : int;
BEGIN
  n := file_ptr(fp)^.pos;
  wsize := n;
  if (file_ptr(fp)^.dirty) and (n <> 0) then
     wsize := write(file_ptr(fp)^.fd,@file_ptr(fp)^.buf,n);
     file_ptr(fp)^.pos := 0;
     file_ptr(fp)^.size := 0;
  end;
  file_ptr(fp)^.dirty := false;
  if n = wsize then
     return 0;
  end;
  return -1;
END;


PROCEDURE fclose(fp: pointer): int;
```

```
   var
     result : int;
  BEGIN
    if fp <> nil then
       result := fflush(fp);
       close(file_ptr(fp)^.fd);
       mem_release(fp);
       return result;
    end;
    return -1;
  END;



  PROCEDURE load_buffer(fp: pointer);
  BEGIN
    file_ptr(fp)^.size :=
read(file_ptr(fp)^.fd,@file_ptr(fp)^.buf,max_buffer);
    file_ptr(fp)^.lpos := file_ptr(fp)^.lpos +
long(file_ptr(fp)^.pos);
    file_ptr(fp)^.pos := 0;
  END;



  { Save file buffer, return zero if ok }
  PROCEDURE save_buffer(fp: pointer): int;
   var
     n : int;
```

```
  BEGIN
    n := file_ptr(fp)^.pos;
    file_ptr(fp)^.lpos := file_ptr(fp)^.lpos + long(n);
    file_ptr(fp)^.pos := 0;
    file_ptr(fp)^.dirty := false;
    if write(file_ptr(fp)^.fd,@file_ptr(fp)^.buf,n) <> n then
       return 1;
    end;
    return 0;
  END;



  PROCEDURE fputc(c: int; fp: pointer): int;
  BEGIN
    file_ptr(fp)^.dirty := true;
    file_ptr(fp)^.buf[file_ptr(fp)^.pos] := char(c);
    inc(file_ptr(fp)^.pos);
    if file_ptr(fp)^.pos >= max_buffer then
       {save_buffer(fp);}
       if save_buffer(fp) <> 0 then
          c := -1;
       end;
{
       if
write(file_ptr(fp)^.fd,@file_ptr(fp)^.buf,file_ptr(fp)^.pos)
<> file_ptr(fp)^.pos then
```

```
          c := -1;

       end;

       file_ptr(fp)^.lpos := file_ptr(fp)^.lpos +
long(file_ptr(fp)^.pos);

       file_ptr(fp)^.pos := 0;

       file_ptr(fp)^.dirty := false;

}

    end;

    return c;

  END;




  PROCEDURE fgetc(fp: pointer): int;

  BEGIN

    if file_ptr(fp)^.pos >= file_ptr(fp)^.size then

       load_buffer(fp);

    end;

    if file_ptr(fp)^.pos < file_ptr(fp)^.size then

       inc(file_ptr(fp)^.pos);

       return int(file_ptr(fp)^.buf[file_ptr(fp)^.pos - 1]);

    end;

    return -1;

  END;




  PROCEDURE fseek(fp: pointer; offset: long; whence: int):
int;
```

```
  BEGIN

    fflush(fp);

    file_ptr(fp)^.lpos :=
lseek(file_ptr(fp)^.fd,offset,seek_set);

    file_ptr(fp)^.pos := 0;

    file_ptr(fp)^.size := 0;

    if file_ptr(fp)^.lpos <> $ffffffff then

       return 0;

    else

       return -1;

    end;

  END;




  PROCEDURE ftell(fp: pointer): long;

  BEGIN

    return file_ptr(fp)^.lpos + long(file_ptr(fp)^.pos);

  END;




  PROCEDURE fread(buf: pointer; size,items: int; fp: pointer):
int;

   var

     n,bufsize,dsize,bpos : int;

  BEGIN

    bpos := 0;
```

```
    dsize := size * items;
    while bpos < dsize loop
      bufsize := file_ptr(fp)^.size - file_ptr(fp)^.pos;
      n := dsize - bpos;
      if n <= bufsize then

copy(@file_ptr(fp)^.buf[file_ptr(fp)^.pos],@data_ptr(buf)^[bpo
s],n);
         file_ptr(fp)^.pos := file_ptr(fp)^.pos + n;
         return items;
      else

copy(@file_ptr(fp)^.buf[file_ptr(fp)^.pos],@data_ptr(buf)^[bpo
s],bufsize);
         file_ptr(fp)^.pos := file_ptr(fp)^.pos + bufsize;
         bpos := bpos + bufsize;
      end;


      if file_ptr(fp)^.pos >= file_ptr(fp)^.size then
         { Read data from file }
         load_buffer(fp);
      end;


      if file_ptr(fp)^.size = 0 then
         { End of file }
         if size < 2 then
            return bpos;
         end;
```

```
          return bpos / size;
        end;
      end loop;
      return items;
    END;


    PROCEDURE fwrite(buf: pointer; size,items: int; fp:
pointer): int;
     var
       n,bufsize,dsize,bpos : int;
    BEGIN
      bpos := 0;
      dsize := size * items;
      while bpos < dsize loop
        bufsize := max_buffer - file_ptr(fp)^.pos;
        n := dsize - bpos;
        if n <= bufsize then

copy(@data_ptr(buf)^[bpos],@file_ptr(fp)^.buf[file_ptr(fp)^.po
s],n);
          file_ptr(fp)^.pos := file_ptr(fp)^.pos + n;
          return items;
        else

copy(@data_ptr(buf)^[bpos],@file_ptr(fp)^.buf[file_ptr(fp)^.po
s],bufsize);
          file_ptr(fp)^.pos := file_ptr(fp)^.pos + bufsize;
```

```
        bpos := bpos + bufsize;
     end;

     if file_ptr(fp)^.pos >= max_buffer then
        { Write data to file }

        if save_buffer(fp) <> 0 then
           { End of file }
           if size < 2 then
              return bpos;
           end;
           return bpos / size;
        end;
     end;
  end loop;
  return items;
END;



{
PROCEDURE fgetc(fp: pointer): int;
 var
   pos,size : int;
BEGIN
  pos := file_ptr(fp)^.pos;
```

```
    size := file_ptr(fp)^.size;
    if pos >= size then
       file_ptr(fp)^.size :=
read(file_ptr(fp)^.fd,@file_ptr(fp)^.buf,max_buffer);
       file_ptr(fp)^.pos := 0;
       pos := 0;
       size := file_ptr(fp)^.size;
    end;
    if pos < size then
       inc(file_ptr(fp)^.pos);
       return int(file_ptr(fp)^.buf[pos]);
    end;
    return -1;
  END;



  var
    fp         : pointer;
    n,t0,t1    : int;
    ch,i       : int;
    buffer     : array[8192] of char;



BEGIN
  if argc() = 2 then
     fp := fopen(argv(1)^,"r");
```

```
    if fp = nil then
       putln(argv(1)^,": couldn't open.");
    else
       putln("rtest: reading """,argv(1)^,""" ...");
       t0 := sys_time();
       n := 0;
 {
       ch := fgetc(fp);
       while (n < sizeof(buffer)) and (ch <> -1) loop
         buffer[n] := char(ch);
         inc(n);
         ch := fgetc(fp);
       end loop;
}
       n := fread(@buffer,1,sizeof(buffer),fp);
       t1 := sys_time();
       fclose(fp);

       fp := fdopen(1);
       if fp <> nil then
          for i := 0 while i < 511 do inc(i) loop
             fputc(int(buffer[i]),fp);
          end loop;
          hold(2000);
          putln("fflush(fp)...");
          fflush(fp);
```

```
        end;


        putln("rtest(stdio): read ",n," bytes took ",t1-t0,"
ms, (",n/((t1-t0)/1000)," bytes/s).");

     end;

  else

     putln("usage: rtest file");

  end;

  }


BEGIN

END stdio;
```

```
STDLIB.TML
{*
 *  stdlib.tml
 *  $Id: stdlib.tml 1.2 1999/05/04 01:28:41 frago Exp frago $
 *
 *  Kernel services standard library for OS-X 2.0
 *
 *  Revision history:
 *    990501 Francis Gormarker, initial issue.
 *
 *    990503 FrGo, renamed unlink() to delete().
 *}

UNIT stdlib;
INTERFACE

  PROCEDURE proc_kill(id, signal: int): int;
  PROCEDURE mkdir(path: string; mode: int): int;
  PROCEDURE chdir(path: string): int;
  PROCEDURE delete(path: string): int;
  PROCEDURE stat(path: string; attr: pointer): int;
  PROCEDURE chmod(path: string; mode: int): int;

IMPLEMENTATION

  uses
```

```
    sys,

    sys_rpc,

    memory,

    file_rpc;




  PROCEDURE kproc_rpc(port: int; service: char; msg:
^kproc_message; size: int): int;

  BEGIN

    msg^.cthread := thread_self();

    return
rpc_call(mod_procd,service,msg,nil,port,size,0,0,5000);

  END;




  PROCEDURE mem_rpc(port: int; service: char; msg: pointer;
size: int): int;

  BEGIN

    return
rpc_call(mod_memd,service,msg,nil,port,size,0,0,5000);

  END;




  PROCEDURE root_rpc(port: int; service: char; msg:
^kproc_message; size: int): int;

  BEGIN

    msg^.cthread := thread_self();

    return
rpc_call(mod_rootd,service,msg,nil,port,size,0,0,mq_infinite);
```

```
  END;




  {
  PROCEDURE _mem_ptr_assign(p: ^int; value: int);
  BEGIN
    if p <> nil then
       p^ := value;
    end;
  END;




  {*
   *  Return zero if ok, return values through int pointer
arguments,
   *  nil pointers are allowed.
   *}
  PROCEDURE mem_avail(map: char; mtotal,mmax,smtotal,smmax:
^int): int;
   var
     msg : mem_avail_msg;
  BEGIN
     msg.id := 0; {get_current()^.proc^.id;}
     msg.map := map;
     if mem_rpc(kernel_port,mem_avail_req,@msg,sizeof(msg)) =
0 then
        _mem_ptr_assign(mtotal,msg.mtotal);
```

```
        _mem_ptr_assign(mmax,msg.mmax);
        _mem_ptr_assign(smtotal,msg.smtotal);
        _mem_ptr_assign(smmax,msg.smmax);
        return 0;
     end;
     return -1;
  END;



  PROCEDURE thread_stat(id,proc_id: int; stat: pointer;
next,next_proc: ^int): int;
   var
     sp  : ^thread_stat_t;
     msg : kproc_message;
  BEGIN
    msg.id := id;
    msg.status := proc_id;
    if kproc_rpc(kernel_port,
                         kproc_thr_stat,
                         @msg,
                         sizeof(kproc_message2) +
sizeof(thread_stat_t)) = 0 then
       copy(@msg.data,stat,sizeof(thread_stat_t));
       if next <> nil then
          next^ := msg.id;
       end;
```

```
        if next_proc <> nil then
           next_proc^ := msg.status;
        end;
        return 0;
     end;
     if next <> nil then
        next^ := -1;
     end;
     if next_proc <> nil then
        next_proc^ := -1;
     end;
     return -1;
   END;
   }



   PROCEDURE proc_kill(id, signal: int): int;
    var
      msg : kproc_message2;
   BEGIN
      msg.id := id;
      msg.status := get_uid();
      msg.size := signal;
      return
kproc_rpc(kernel_port,kproc_kill,@msg,sizeof(kproc_message2));
   END;
```

```
  PROCEDURE mkdir(path: string; mode: int): int;

   var

     msg     : kproc_message;

  BEGIN

     msg.data := path;

     msg.status := mode;

     return
root_rpc(kernel_port,kproc_mkdir,@msg,sizeof(kproc_message2) +
length(msg.data) + 1);

  END;




  PROCEDURE chdir(path: string): int;

   var

     msg     : kproc_message;

  BEGIN

     msg.data := path;

     return
root_rpc(kernel_port,kproc_chdir,@msg,sizeof(kproc_message2) +
length(msg.data) + 1);

  END;




  PROCEDURE delete(path: string): int;

   var

     msg     : kproc_message;
```

```
  BEGIN

     msg.data := path;

     return
root_rpc(kernel_port,kproc_delete,@msg,sizeof(kproc_message2)
+ length(msg.data) + 1);

  END;




  PROCEDURE stat(path: string; attr: pointer): int;

   var

     n       : int;

     info    : ^lookup_info;

     ufid    : ufid_t;

     msg     : kproc_message;

  BEGIN

     {putln("stat(",path,")");}

     msg.data := path;

     n := length(msg.data) + 1;

     if n < sizeof(lookup_info) then

        n := sizeof(lookup_info);

     end;

     if
root_rpc(kernel_port,kproc_path_lookup,@msg,sizeof(kproc_messa
ge2) + n) = 0 then

        info := @msg.data;

        ufid := info^.ufid;

        return get_attr(@ufid,attr);
```

```
    end;
    return -1;
  END;


  PROCEDURE chmod(path: string; mode: int): int;
   var
    attr    : fattr_t;
  BEGIN
    if stat(path,@attr) = 0 then
       attr.mode := mode;
       return set_attr(@attr.ufid,@attr);
    end;
    return -1;
  END;


BEGIN
END stdlib;
```

```
SYS_RPC.TML

{*

 *  sys_rpc.tml

 *  $Id: sys_rpc.tml 1.33 2000/04/12 17:44:56 frago Exp frago
$

 *

 *  Definitions for OS-X system RPC

 *

 *  Revision history:

 *    940828 Francis Gormarker, initial OS-X 1.0 version.

 *

 *    980429 FrGo, modified for OS-X 2.0

 *

 *    980529 FrGo, moved process messages and service codes
from kproc.tml.

 *

 *    990125 FrGo, modified GFD type; file position is now 32
bits (long),

 *           added use_count field.

 *

 *    990129 FrGo, added kproc_open, close, dup service codes.

 *

 *    990206 FrGo, increased message size to 288 bytes (256 +
32).

 *

 *    990213 FrGo, changed rpc_header.net_size to 16 bits,
changed file size

 *           and position fields to 32 bits.
```

```
 *
 *    990305 FrGo, added proper message types and service IDs
for memory
 *           server.
 *
 *    990311 FrGo, updated message types and service IDs for
file/dir services.
 *
 *    990313 FrGo, changed numbering of o_read, o_write etc so
that b0=read,
 *           b1=write.
 *
 *    990315 FrGo, added new message structure for
proc_create.
 *
 *    990319 FrGo, added new structure for thread status.
Added kproc_mkdir
 *           and kproc_chdir service codes etc.
 *
 *    990326 FrGo, moved device descriptor type from threads
unit. Added
 *           new semaphore, mutex and UFID fields to dev_desc.
 *
 *    990331 FrGo, updated message structure and service codes
for name server.
 *
 *    990406 FrGo, added fields argc, argv to proc_attr_msg.
 *
 *    990408 FrGo, added standard protocol codes.
```

```
 *
 *    990412 FrGo, added kproc device service codes, file
service codes for
 *           mount, unmount, sync etc. Added directory entry
structure.
 *
 *    990420 FrGo, added file service code file_delete.
 *
 *    990421 FrGo, modified for POSIX style symbolic file mode
constants.
 *
 *    990428 FrGo, added kproc_path_xx services. Added
proc_attr structure.
 *           Moved some IPC constants and IPC buffer types
from threads unit.
 *           Added process fields to thread_attr structure.
 *
 *    990502 FrGo, added kproc_get_pstat, kproc_set_pstat,
kproc_get_root,
 *           kproc_set_root, kproc_get_device services.
 *           Added proc_stat_t structure.
 *
 *    990515 FrGo, added file_sys_msg structure.
 *
 *    990602 FrGo, bugfix: data field of rpc_message type was
only 256 bytes,
 *           now increased to 274 bytes.
 *
 *    990614 FrGo, added flags, usr1, usr2 fields to dev_desc
structure. Added
```

```
 *           file_ioctl service code and file_ioctl_msg
structure.
 *
 *    990615 FrGo, file_close, file_open service codes.
 *
 *    991105 FrGo, modified name server message type.
 *
 *    991212 FrGo, added file_ioctl_msg2;
 *
 *    000331 FrGo, added file mode field to GFD descriptor.
 *
 *    000412 FrGo, added kproc_ufid_lookup and kproc_ufid_bind
service codes.
 *}


UNIT sys_rpc;


INTERFACE


  const

    net_port          = 0;

    kernel_port       = 1;

    dir_port          = 2;

    name_port         = 3;

    max_file_name     = 32;

    max_file_data     = 256;
```

```
max_ioctl_data    = 128;
dir_max_name      = 24;
dir_entry_size    = 32;
max_name_len      = 16;
{max_name_bind     = 16;}
max_name_data     = 128;
max_net_data      = 249;
max_arg           = 16;
max_arg_size      = 180;
max_mem_entries   = 64;
max_message       = 288;

o_read            = 1;
o_write           = 2;
o_rdwr            = 3;
rd_nowait         = 32768;
ipc_nowait        = 65535;
rpc_timeout       = 1000;

exit_normal       = $0000;   { Standard exit codes }
exit_killed       = $1000;
exit_stopped      = $2000;
exit_syserr       = $f000;

                          { System error codes }
e_io              = 1;   { Read / write error }
```

```
    e_nospace         = 2;   { File system full }
    e_ftab            = 3;   { File system tables full }
    e_format          = 4;   { Bad file system }
    e_pos             = 5;   { Bad file position }
    e_path            = 6;   { Bad file path }
    e_perm            = 7;   { Permission denied }
    e_arg             = 8;   { Bad argument to system call }
    e_intern          = 9;   { System internal error }
    e_rpc             = 10;  { RPC error / timeout / no reply
}
    e_access          = 11;  { File access denied }
    e_notfound        = 12;  { File not found }
    e_notdir          = 13;  { File is not a directory }
    e_exist           = 14;  { File already exists }
    e_route           = 15;  { No route to destination / bad
port address }
    e_service         = 16;  { Unknown RPC service or module
code }


    mq_any        = 65535;   { Message queue special codes }
    mq_infinite   = 65535;


    thread_free   = #0;      { Thread status codes }
    thread_ready  = #1;
    thread_wait   = #2;
    thread_term   = #3;
    thread_susp   = #4;
```

```
thread_hold   = #5;

seek_set          = 0;   { Whence codes for lseek }
seek_cur          = 1;

io_null           = 0;      { ioctl command codes }
io_get_info       = 1;
io_get_mode       = 2;
io_set_mode       = 3;
io_get_speed      = 4;
io_set_speed      = 5;
io_get_win        = 6;
io_set_win        = 7;

io_mode_break     = 1;      { ioctl mode flags }
io_mode_eof       = 2;
io_mode_nblock    = 4;

io_speed_9600     = 960;    { ioctl speed constants }
io_speed_19200    = 1920;
io_speed_38400    = 3840;
io_speed_31250    = 3125;


s_ifmt            = $f000;  { File type mask }
```

```
    s_iamb          = $0fff;  { Access mode mask }

    s_ifreg         = $0000;  { Regular file }

    s_ififo         = $b000;  { FIFO special file or pipe }

    s_ifchr         = $9000;  { Character special file }

    s_ifdir         = $8000;  { Directory }

    s_ifblk         = $a000;  { Block special file }

    s_iflnk         = $c000;  { Symbolic link }


    s_isuid         = $0800;  { Set user ID on execution }

    s_isgid         = $0080;  { Set group ID on execution }

    s_isvtx         = $0008;  { Save text image after exec }


    s_irwxu         = $0700;  { Read, write, execute by
owner }

    s_irusr         = $0400;  { Read by owner }

    s_iwusr         = $0200;  { Write by owner }

    s_ixusr         = $0100;  { Execute (search dir) by
owner }


    s_irwxg         = $0070;  { Read, write, execute by
group }

    s_irgrp         = $0040;  { Read by group }

    s_iwgrp         = $0020;  { Write by group }

    s_ixgrp         = $0010;  { Execute (search dir) by
group }


    s_irwxo         = $0007;  { Read, write, execute by
others }
```

```
    s_iroth           = $0004;  { Read by others }

    s_iwoth           = $0002;  { Write by others }

    s_ixoth           = $0001;  { Execute (search dir) by
others }


                              { Process creation flags }

    pflags_default    = 0;   { Default: clear memory, kill
enabled, separate mapping }

    pflags_no_clear   = 1;   { Don't clear program segment }

    pflags_no_kill    = 2;   { Disable TTY kill }

    pflags_packed     = 4;   { Pack process, don't require
separate mapping }


    mod_dird          = #1;

    mod_filed         = #2;

    mod_memd          = #8;   { Kernel module codes }

    mod_procd         = #9;

    mod_named         = #10;

    mod_rootd         = #11;

    mod_execd         = #12;


    proto_user        = #0;   { Protocol codes }

    proto_rpc         = #1;

    proto_rpc_reply   = #2;


    std_null          = #0;   { Standard service codes }

    std_ver           = #1;
```

```
std_shutdown      = #2;
std_restart       = #3;

ftype_reg         = #$00;
ftype_dir         = #$10;
ftype_stream      = #$20;
ftype_block       = #$30;

mem_alloc_req     = #8;   { Memory server service codes }
mem_release_req   = #9;
mem_sm_alloc      = #10;
mem_sm_release    = #11;
mem_avail_req     = #12;
mem_info_req      = #13;

kproc_thr_create   = #1;  { Process server service codes }
kproc_thr_kill     = #2;
kproc_thr_join     = #3;
kproc_thr_wait     = #4;
kproc_thr_stat     = #5;
kproc_thr_exit     = #6;
kproc_create       = #7;
kproc_read         = #8;
kproc_write        = #9;
kproc_start        = #10;
kproc_exit         = #11;
```

```
kproc_wait          = #12;
kproc_kill          = #13;
kproc_alloc         = #14;
kproc_release       = #15;
kproc_get_pstat     = #16;
kproc_set_pstat     = #17;
kproc_tty_kill      = #18;

kproc_open          = #32;
kproc_close         = #33;
kproc_dup           = #34;
kproc_mkdir         = #35;
kproc_chdir         = #36;
kproc_stat          = #37;
kproc_chmod         = #38;
kproc_delete        = #39;
kproc_rename        = #40;
kproc_add_device    = #41;
kproc_del_device    = #42;
kproc_path_lookup   = #43;
kproc_path_create   = #44;
kproc_path_write    = #45;
kproc_path_link     = #46;
kproc_path_unlink   = #47;
kproc_get_device    = #48;
```

```
    kproc_get_root     = #49;
    kproc_set_root     = #50;
    kproc_ufid_open    = #51;
    kproc_ufid_lookup  = #52;
    kproc_ufid_bind    = #53;

    root_open          = #16; { Root server service codes }
    root_close         = #17;
    root_dup           = #18;
    root_bind          = #19;
    root_net_info      = #20;

    link_test          = #1;  { Data link test message codes }
    link_reply         = #2;


    dir_lookup         = #8;  { Directory server service codes
}
    dir_insert         = #9;
    dir_delete         = #10;
    dir_rlookup        = #11;
    dir_rename         = #12;

    file_read          = #8;  { File server service codes }
    file_write         = #9;
    file_create        = #10;
```

```
  file_get_attr     = #11;
  file_set_attr     = #12;
  file_ioctl        = #13;
  file_delete       = #14;
  file_mount        = #15;
  file_unmount      = #16;
  file_sync         = #17;
  file_sys_stat     = #18;
  file_close        = #19;
  file_open         = #20;

  name_lookup_req       = #8;  { Name server service codes }
  name_get_domains_req  = #9;
  name_get_keys_req     = #10;
  name_bind_req         = #11;
  name_unbind_req       = #12;
  name_rlookup_req      = #13;
  name_create_req       = #14;
  name_delete_req       = #15;

  exec_exec         = #8;  { Exec server service codes }


type
  ufid_t = record
             server : int;
```

```
            dev    : char;
            flags  : char;
            id     : int;
          end;

    ufid_ptr = ^ufid_t;

    gfd_t = record
            pos       : long;
            ufid      : ufid_t;
            read      : pointer;
            write     : pointer;
            ioctl     : pointer;
            dev       : pointer;
            owner     : int;
            use_count : int;
            mode      : int;
          end;

    gfd_ptr = ^gfd_t;

    ipc_buffer = record
                   next        : pointer;
                   prev        : pointer;
                   prio        : char;       { Priority of
user thread }
```

```
                    size        : int;        { Current data
size }
                    src         : int;        { ID of source
thread }
                    owner       : int;        { Thread ID of
owner }
                    bufsize     : int;        { Buffer max size
}
                    data        : array[max_message] of char;
                  end;

    ipc_buf_ptr = ^ipc_buffer;

    dir_name = array[dir_max_name] of char;

    dir_entry = record
                  size  : int;
                  node  : ufid_t;
                  name  : dir_name;
                end;

    tname_string = array[16] of char;

    thread_attr = record
                    stackaddr   : pointer;
                    stacksize   : int;
                    priority    : int;
```

```
                detachstate : int;
                name        : tname_string;
              end;

thread_attr_ptr = ^thread_attr;

fd_attr =     record
                index    : int;
                pos      : long;
                ufid     : ufid_t;
              end;

fd_attr_array = array[8] of fd_attr;

proc_attr =   record
                map      : int;
                parent   : int;
                uid      : int;
                tty      : int;
                sigmask  : int;
                cwd      : ufid_t;
                user1    : int;
                user2    : int;
                user3    : int;
                user4    : int;
              end;
```

```
proc_attr_ptr = ^proc_attr;


hw_info = record
            read   : pointer;
            write  : pointer;
            ioctl  : pointer;
            dtype  : char;
            base   : char;
            vector : char;
            name   : array[16] of char;
          end;



devname_string = array[8] of char;


dev_desc =    record
                id           : int;
                rx_buf       : pointer;
                tx_buf       : pointer;
                rx_sem       : pointer;
                tx_sem       : pointer;
                mutex        : pointer;
                status_addr  : int;
                ctrl_addr    : int;
```

```
                rxd_addr     : int;
                txd_addr     : int;
                usr1_addr    : int;
                usr2_addr    : int;
                rx_vector    : int;
                tx_vector    : int;
                flags        : int;
                mode         : int;
                usr1         : int;
                usr2         : int;
                ufid         : ufid_t;
                read         : pointer;
                write        : pointer;
                ioctl        : pointer;
                name         : devname_string;
              end;

dev_ptr = ^dev_desc;

data_fn = procedure(gfd,buf: pointer; size: int): int;
{list_fn = procedure(env,item: pointer);}


fattr_t = record
            mode   : int;
            uid    : int;
```

```
            size   : long;
            time   : long;
            server : int;
            ufid   : ufid_t;
            spare1 : int;
            spare2 : int;
          end;

stat_t = record
           mode   : int;
           uid    : int;
           size   : long;
           time   : long;
           spare  : int;
           ufid   : ufid_t;
           pos    : int;
           posHi  : int;
         end;

fs_info_t = record
              fs_type     : int;
              blocks      : int;
              nodes       : int;
              free_blocks : int;
              free_nodes  : int;
```

```
                end;


    hType  = record
                checksum   : int;  {  16 bit checksum  }
                magicNum   : int;
                codeOffs   : int;  {  File offset for code  }
                codeSize   : int;  {  Size in bytes  }
                dataSize   : int;  {  DSEG size, stack & heap
excl  }
                stackSize  : int;  {  Main unit stack size  }
                heapSize   : int;  {  Desired heap size  }
                symOffs    : int;  {  File offset for symbol
table  }
                symSize    : int;  {  Number of symbols  }
                symExport  : int;  {  First exported symbol
index  }
                symImport  : int;  {  First imported symbol
index  }
                symPrivate : int;  {  First private symbol
index   }
                strOffs    : int;  {  File offset for string
table  }
                strSize    : int;  {  Size in bytes
}
                strExport  : int;  {  First exported id index
}
                strImport  : int;  {  First imported id index
}
                strPrivate : int;  {  First private id index
}
```

```
                crelOffs   : int;  {  File offset for CSEG
reloc-info  }
                crelSize   : int;  {  Number of records  }
                drelOffs   : int;  {  File offset for DSEG
reloc-info  }
                drelSize   : int;  {  Number of records  }
                debugOffs  : int;  {  File offset for debug
info  }
                debugSize  : int;  {  Number of records  }
              end;


    thread_stat_t = record
                id         : int;       { Thread index }
                prio       : char;      { Thread priority,
255=highest, 0=lowest }
                s_prio     : char;      { Static priority }
                status     : char;      { Current status }
                map        : char;      { Process page mapping
number }
                sp         : int;       { Current SP-reg save }
                start      : pointer;   { Thread entry address }
                proc_id    : int;       { Enclosing process }
                mq         : int;       { Default message queue
for RPCs }
                error      : int;       { System error code }
                sseg       : pointer;   { Stack segment start }
                ssize      : int;       { Stack segment size }
                arg        : pointer;   { Thread user argument }
```

```
                exitcode  : int;        { Termination status
code }

                wait_id   : int;        { ID of thread or
process to wait for }

                proc_stk  : boolean;    { True if stack owned by
process }

                detached  : boolean;    { True if thread is
detached, unjoinable }

                delay     : int;        { Thread delay time }

                delay_t0  : int;        { Thread delay start
time }

                parent    : int;        { Parent of enclosing
process }

                pseg      : int;        { Process segment base }

                psize     : int;        { Process segment size }

                tty       : int;        { Associated terminal }

                uid       : int;        { User ID }

                name      : tname_string;  { Thread name }

                pname     : tname_string;  { Name of enclosing
process }

    end;


    proc_stat_t = record

                id        : int;        { Process index }

                status    : char;       { Current status }

                map       : char;       { Process page mapping
number }

                uid       : int;        { User ID }

                tty       : int;        { Associated terminal }

                pseg      : int;        { Process segment base }
```

```
                psize     : int;        { Process segment size }
                parent    : int;        { Parent of enclosing
process }
                threads   : int;        { No. of threads in
process }
                user1     : int;
                user2     : int;
                name      : tname_string;  { Process name }
    end;


    net_message = record
                    net_src  : int;
                    net_dst  : int;
                    net_size : char;
                    protocol : char;
                    data     : array[max_net_data] of char;
                  end;


   net_packet = record
                  start    : char;   { Start byte }
                  check    : char;   { Header checksum }
                  datchk   : char;   { Data checksum }
                  status   : char;   { Ack flag & sequence no
}
                  source   : int;    { Source socket }
                  dest     : int;    { Destination socket }
                  size     : char;   { User data size }
```

```
                protocol : char;
                data     : array[max_net_data] of char;
              end;


    rpc_header = record
                  net_src  : int;
                  net_dst  : int;
                  net_size : int;
                  protocol : char;
                  module   : char;
                  service  : char;
                  spare    : char;
                  seq      : int;
                  result   : int;
                end;

    rpc_message = record
                   header   : rpc_header;
                   data     : array[274] of char;
                 end;

    mem_alloc_msg = record
                      header    : rpc_header;
                      id        : int;          { Process ID
of caller }
```

```
                    p          : pointer;
                    size       : int;
                  end;

    mem_avail_msg = record
                    header     : rpc_header;
                    id         : int;          { Process ID
of caller }
                    map        : char;         { Memory map
no. }
                    mtotal     : int;          { Total
available in map }
                    mmax       : int;          { Largest
block in map }
                    smtotal    : int;          { Total
available SM }
                    smmax      : int;          { Largest
block in SM }
                  end;

    mem_entry =    record
                    base : int;
                    size : int;
                  end;

    mem_info_msg = record
                    header     : rpc_header;
                    id         : int;          { Process ID of
caller }
```

```
                    map       : char;          { Memory map
no. }
                    get_used  : char;          { Flag, non-
zero => get used blocks }
                    entries   : int;           { No of entries
in block list }
                    entry     : array[max_mem_entries] of
mem_entry;
                  end;


    proc_message = record
                    header    : rpc_header;
                    id        : int;
                    size      : int;
                    data      : array[200] of char;
                  end;


    proc_message2 = record
                     header    : rpc_header;
                     id        : int;
                     size      : int;
                    end;


    kproc_message = record
                     header    : rpc_header;
                     cthread   : int;
                     id        : int;
```

```
                status    : int;
                size      : int;
                func      : pointer;
                arg       : pointer;
                data      : array[128] of char;
              end;

kproc_message2 = record
                 header    : rpc_header;
                 cthread   : int;
                 id        : int;
                 status    : int;
                 size      : int;
                 func      : pointer;
                 arg       : pointer;
               end;

file_name_string = array[max_file_name] of char;

lookup_info = record
              ufid : ufid_t;
              dir  : ufid_t;
              last : file_name_string;
            end;
```

```
proc_attr_msg = record
                  header    : rpc_header;
                  cpid      : int;
                  id        : int;
                  size      : int;
                  flags     : int;
                  uid       : int;
                  argc      : int;
                  argv      : int;
                  ufid      : ufid_t;
                  tattr     : thread_attr;
                  name      : tname_string;
                  cwd       : ufid_t;
                  files     : int;
                  fd        : fd_attr_array;
                end;


dir_message = record
                header   : rpc_header;
                uid      : int;
                mode     : int;
                index    : int;
                dir      : ufid_t;
                ufid     : ufid_t;
```

```
                name     : file_name_string;
                data     : array[128] of char;
              end;


dir_message2 = record
                header   : rpc_header;
                uid      : int;
                mode     : int;
                index    : int;
                dir      : ufid_t;
                ufid     : ufid_t;
                name     : file_name_string;
              end;


file_message = record
                header   : rpc_header;
                uid      : int;
                pos      : long;
                size     : int;
                ufid     : ufid_t;
                data     : array[max_file_data] of char;
              end;
```

```
file_message2 = record
                  header   : rpc_header;
                  uid      : int;
                  pos      : long;
                  size     : int;
                  ufid     : ufid_t;
                end;


file_attr_msg = record
                  header   : rpc_header;
                  uid      : int;
                  ufid     : ufid_t;
                  attr     : fattr_t;
                end;


file_sys_info = record
                  ufid        : ufid_t;
                  magic       : int;
                  fs_type     : int;
                  block_size  : int;
                  blocks      : long;
                  cluster     : int;
                  nodes       : int;
```

```
                dev         : ufid_t;
                free_blocks : long;
                free_nodes  : int;
              end;


file_sys_msg = record
                header      : rpc_header;
                ufid        : ufid_t;
                info        : file_sys_info;
              end;


file_ioctl_msg2 = record
                   header   : rpc_header;
                   uid      : int;
                   ufid     : ufid_t;
                   cmd      : int;
                   size     : int;
                 end;


file_ioctl_msg = record
                  header   : rpc_header;
                  uid      : int;
                  ufid     : ufid_t;
```

```
                        cmd      : int;
                        size     : int;
                        data     : array[max_ioctl_data] of
char;
                      end;


    name_string = array[max_name_len] of char;


    name_message = record
                     header  : rpc_header;
                     uid     : int;
                     cpid    : int;
                     flags   : int;
                     size    : int;
                     value   : int;
                     spare   : int;
                     index   : int;
                     domain  : name_string;
                     key     : name_string;
                     data    : array[max_name_data] of char;
                   end;


    name_message2 = record
                      header  : rpc_header;
                      uid     : int;
```

```
                cpid    : int;
                flags   : int;
                size    : int;
                value   : int;
                spare   : int;
                index   : int;
                domain  : name_string;
                key     : name_string;
            end;

arg_array = array[max_arg] of ^string;
arg_ptr   = ^arg_array;

exec_message = record
                header : rpc_header;
                id     : int;
                tty    : int;
                pri    : int;
                stdin  : ufid_t;
                stdout : ufid_t;
                stderr : ufid_t;
                cwd    : array[16] of char;
                argc   : int;
                argv   : array[max_arg_size] of char;
            end;
```

```
    link_message = record
                     header : rpc_header;
                     state  : int;
                     host   : int;
                     name   : array[16] of char;
                   end;


IMPLEMENTATION


BEGIN
END sys_rpc;
```

```
SYS.TML
{*
 *  sys.tml
 *  $Id: sys.tml 1.16 1999/11/11 19:05:52 frago Exp frago $
 *
 *  OS-X 2.0 system call entry point unit.
 *
 *  Revision history:
 *    990125 Francis Gormarker, initial issue.
 *
 *    990129 FrGo, added dup().
 *
 *    990311 FrGo, added ufid_lookup, ufid_bind, _int_signal,
rpc_call.
 *
 *    990312 FrGo, bugfix: bad entry addresses for ufid_bind &
ufid_lookup.
 *
 *    990315 FrGo, updated with new proc_create() interface.
 *
 *    990319 FrGo, updated with new process functions.
 *
 *    990402 FrGo, added exit() which calls proc_exit().
 *
 *    990406 FrGo, bugfix: bad entry addresses for sys_time,
thread_exit,
 *           thread_self,mq_self. Added thread_arg(), argc(),
argv() functions.
```

```
 *           Added process arguments to proc_create().
 *
 *    990411 FrGo, added ufid_read, ufid_write, shmem_alloc,
shmem_release.
 *
 *    990412 FrGo, added phys_read, phys_write.
 *
 *    990423 FrGo, Added some vital entry points in threads
unit
 *           (sync_alloc, sem_init etc). Completely rearranged
entry
 *           point table.
 *
 *    990425 FrGo, added entry points for thread_get_gfd and
lseek.
 *
 *    990428 FrGo, removed entry points for get_sigmask,
set_sigmask.
 *
 *    990502 FrGo, added entry points for get_sigmask,
set_sigmask again.
 *           Added get_time, set_time, get_device.
 *
 *    991110 FrGo, removed get_hostname() and set_hostname()
entries, added
 *           proc_self() entry point.
 *}


UNIT sys;
```

```
INTERFACE

  { From threads.tml }
  PROCEDURE hold(time: int);
  PROCEDURE lock();
  PROCEDURE unlock();
  PROCEDURE sleep();

  PROCEDURE int_enter;
  PROCEDURE int_exit;
  PROCEDURE _int_cause;
  PROCEDURE _int_signal;

  PROCEDURE await(q: pointer);
  PROCEDURE cause(q: pointer);
  PROCEDURE sem_wait(s: pointer);
  PROCEDURE sem_trywait(s: pointer): int;
  PROCEDURE sem_getvalue(s: pointer): int;
  PROCEDURE sem_signal(s: pointer);
  PROCEDURE mutex_wait(s: pointer);
  PROCEDURE mutex_signal(s: pointer);

  PROCEDURE sys_time(): int;
  PROCEDURE thread_exit(status: int);
  PROCEDURE thread_self(): int;
```

```
PROCEDURE mq_self(): int;

PROCEDURE thread_arg(): pointer;
PROCEDURE argc(): int;
PROCEDURE argv(n: int): ^string;

PROCEDURE get_uid(): int;
PROCEDURE set_uid(uid: int);
PROCEDURE get_sigmask(): int;
PROCEDURE set_sigmask(mask: int);

{
PROCEDURE get_proc_attr(attr: pointer);
PROCEDURE set_proc_attr(attr: pointer);
}


PROCEDURE get_syserr(): int;
PROCEDURE set_syserr(error: int);

PROCEDURE sync_alloc(): pointer;
PROCEDURE sync_release(p: pointer);
PROCEDURE sem_init(s: pointer; value: int; limit: int);
PROCEDURE mutex_init(m: pointer);
```

```
  PROCEDURE create_int_wrap(buf: ^string; desc,func: pointer):
int;

  PROCEDURE _int_tty_kill;

  PROCEDURE _tty_kill;

  PROCEDURE thread_get_gfd(fd: int): pointer;



  PROCEDURE mq_create(id: int): int;

  PROCEDURE mq_destroy(id: int);

  PROCEDURE mq_send(src,dst: int; data: pointer; size: int):
int;

  PROCEDURE mq_recv(local: int; var src: int; data: pointer;
size,timeout: int): int;

  PROCEDURE mq_recv_from(local: int; var src: int; data:
pointer; size,timeout: int): int;



  PROCEDURE ipc_buf_alloc(size: int): pointer;

  PROCEDURE ipc_buf_release(p: pointer);




  { From syscalls.tml }

  PROCEDURE get_hostaddr(): int;

  {PROCEDURE get_hostname(): ^string;}

  PROCEDURE set_hostaddr(n: int);

  {PROCEDURE set_hostname(s: string);}



  PROCEDURE thread_create(attr: pointer; func,arg: pointer):
int;

  PROCEDURE thread_join(id: int; var status: int): int;
```

```
  PROCEDURE thread_kill(id, signal: int);

  PROCEDURE rpc_call(module,service: char; msg,buf: pointer;
server,hsize,dsize,rsize,timeout: int): int;


  PROCEDURE proc_create(host  : int;

                        size  : int;

                        flags : int;

                        argc  : int;

                        argv  : int;

                        tattr : pointer;

                        files : int;

                        name  : string;

                        ufid  : pointer): int;


  PROCEDURE proc_self(): int;

  PROCEDURE proc_start(id,addr: int): int;

  PROCEDURE proc_exit(status: int);

  PROCEDURE proc_wait(status: ^int): int;

  {PROCEDURE proc_kill(id, signal: int): int;}

  PROCEDURE exit(status: int);


  PROCEDURE mem_alloc(size: int): pointer;

  PROCEDURE mem_release(p: pointer);

  PROCEDURE shmem_alloc(size: int): pointer;

  PROCEDURE shmem_release(p: pointer);
```

```
  PROCEDURE read(fd: int; buf: pointer; size: int): int;

  PROCEDURE write(fd: int; buf: pointer; size: int): int;



  PROCEDURE open(path: string; mode: int): int;

  PROCEDURE close(fd: int): int;

  PROCEDURE dup(fd: int): int;

  PROCEDURE ufid_lookup(ufid: pointer): int;

  PROCEDURE ufid_bind(ufid: pointer; port: int);

  PROCEDURE ufid_read(ufid: pointer; pos: long; buf: pointer;
size: int): int;

  PROCEDURE ufid_write(ufid: pointer; pos: long; buf: pointer;
size: int): int;

  PROCEDURE phys_read(addr: long): char;

  PROCEDURE phys_write(addr: long; data: char);

  PROCEDURE lseek(fd: int; offset: long; whence: int): long;

  PROCEDURE get_time(): long;

  PROCEDURE set_time(t: long);

  PROCEDURE get_device(id: int): pointer;




IMPLEMENTATION



  PROCEDURE hold(time: int);     extern $fd00;

  PROCEDURE lock();              extern $fd04;

  PROCEDURE unlock();            extern $fd08;

  PROCEDURE sleep();             extern $fd0c;
```

```
PROCEDURE int_enter;           extern $fd10;
PROCEDURE int_exit;            extern $fd14;
PROCEDURE _int_cause;          extern $fd18;
PROCEDURE _int_signal;         extern $fd1c;

PROCEDURE await(q: pointer);                  extern $fd20;
PROCEDURE cause(q: pointer);                  extern $fd24;
PROCEDURE sem_wait(s: pointer);               extern $fd28;
PROCEDURE sem_trywait(s: pointer): int;       extern $fd2c;
PROCEDURE sem_getvalue(s: pointer): int;      extern $fd30;
PROCEDURE sem_signal(s: pointer);             extern $fd34;
PROCEDURE mutex_wait(s: pointer);             extern $fd38;
PROCEDURE mutex_signal(s: pointer);           extern $fd3c;

PROCEDURE sys_time(): int;                    extern $fd40;
PROCEDURE thread_exit(status: int);           extern $fd44;
PROCEDURE thread_self(): int;                 extern $fd48;
PROCEDURE mq_self(): int;                     extern $fd4c;

PROCEDURE thread_arg(): pointer;              extern $fd50;
PROCEDURE argc(): int;                        extern $fd54;
PROCEDURE argv(n: int): ^string;              extern $fd58;

PROCEDURE get_uid(): int;                     extern $fd5c;
PROCEDURE set_uid(uid: int);                  extern $fd60;
```

```
  PROCEDURE get_sigmask(): int;                extern $fd64;

  PROCEDURE set_sigmask(mask: int);            extern $fd68;


  {

  PROCEDURE get_proc_attr(attr: pointer);      extern $fd64;

  PROCEDURE set_proc_attr(attr: pointer);      extern $fd68;

  }


  PROCEDURE get_syserr(): int;                 extern $fd6c;

  PROCEDURE set_syserr(error: int);            extern $fd70;


  PROCEDURE sync_alloc(): pointer;             extern $fd74;

  PROCEDURE sync_release(p: pointer);          extern $fd78;


  PROCEDURE sem_init(s: pointer; value: int; limit: int);
extern $fd7c;

  PROCEDURE mutex_init(m: pointer);
extern $fd80;


  PROCEDURE create_int_wrap(buf: ^string; desc,func: pointer):
int; extern $fd84;


  PROCEDURE _int_tty_kill;                     extern $fd88;

  PROCEDURE _tty_kill;                         extern $fd8c;

  PROCEDURE thread_get_gfd(fd: int): pointer;  extern $fd90;


  PROCEDURE mq_create(id: int): int;
extern $fe00;
```

```
  PROCEDURE mq_destroy(id: int);
extern $fe04;

  PROCEDURE mq_send(src,dst: int; data: pointer; size: int):
int;                            extern $fe08;

  PROCEDURE mq_recv(local: int; var src: int; data: pointer;
size,timeout: int): int;        extern $fe0c;

  PROCEDURE mq_recv_from(local: int; var src: int; data:
pointer; size,timeout: int): int;   extern $fe10;



  PROCEDURE ipc_buf_alloc(size: int): pointer;  extern $fe14;

  PROCEDURE ipc_buf_release(p: pointer);        extern $fe18;




  { From syscalls.tml }

  PROCEDURE get_hostaddr(): int;               extern $fe40;

  {PROCEDURE get_hostname(): ^string;           extern $fe44;}

  PROCEDURE set_hostaddr(n: int);              extern $fe48;

  {PROCEDURE set_hostname(s: string);           extern $fe4c;}



  PROCEDURE thread_create(attr: pointer; func,arg: pointer):
int;  extern $fe50;

  PROCEDURE thread_join(id: int; var status: int): int;
extern $fe54;

  PROCEDURE thread_kill(id, signal: int);
extern $fe58;

  PROCEDURE rpc_call(module,service: char; msg,buf: pointer;
server,hsize,dsize,rsize,timeout: int): int; extern $fe5c;



  PROCEDURE proc_create(host  : int;
```

```
                        size  : int;
                        flags : int;
                        argc  : int;
                        argv  : int;
                        tattr : pointer;
                        files : int;
                        name  : string;
                        ufid  : pointer): int;
extern $fe60;


  PROCEDURE proc_self(): int;
extern $fe64;


  PROCEDURE proc_start(id,addr: int): int;
extern $fe6c;

  PROCEDURE proc_exit(status: int);
extern $fe70;

  PROCEDURE proc_wait(status: ^int): int;
extern $fe74;

  {PROCEDURE proc_kill(id, signal: int): int;
extern $fe78;}


  PROCEDURE mem_alloc(size: int): pointer;
extern $fe80;

  PROCEDURE mem_release(p: pointer);
extern $fe84;

  PROCEDURE shmem_alloc(size: int): pointer;
extern $fe88;

  PROCEDURE shmem_release(p: pointer);
extern $fe8c;
```

```
  PROCEDURE read(fd: int; buf: pointer; size: int): int;
extern $fea0;

  PROCEDURE write(fd: int; buf: pointer; size: int): int;
extern $fea4;



  PROCEDURE open(path: string; mode: int): int;
extern $fea8;

  PROCEDURE close(fd: int): int;
extern $feac;

  PROCEDURE dup(fd: int): int;
extern $feb0;

  PROCEDURE ufid_lookup(ufid: pointer): int;
extern $feb4;

  PROCEDURE ufid_bind(ufid: pointer; port: int);
extern $feb8;

  PROCEDURE ufid_read(ufid: pointer; pos: long; buf: pointer;
size: int): int;  extern $febc;

  PROCEDURE ufid_write(ufid: pointer; pos: long; buf: pointer;
size: int): int; extern $fec0;

  PROCEDURE phys_read(addr: long): char;
extern $fec4;

  PROCEDURE phys_write(addr: long; data: char);
extern $fec8;

  PROCEDURE lseek(fd: int; offset: long; whence: int): long;
extern $fecc;

  PROCEDURE get_time(): long;
extern $fed0;

  PROCEDURE set_time(t: long);
extern $fed4;

  PROCEDURE get_device(id: int): pointer;
extern $fed8;
```

```
  PROCEDURE exit(status: int);
  BEGIN
    proc_exit(status);
  END;



BEGIN
END sys;
```