```
#!/bin/bash
#
# mklibs.sh: An automated way to create a minimal /lib/ directory.
#
# Copyright 1999 by Marcus Brinkmann <Marcus.Brinkmann@ruhr-uni-bochum.de>
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 2 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

# STATIC DATA SECTION
#

usage="Usage: $0 [OPTION]... -d DEST FILE ..."
try="Try "\`"$0 --help' for more information"
version="$0 0.1, Copyright 1999 Marcus Brinkmann"

PATH=/bin:/usr/bin

default_src_path=/lib:/usr/lib
dest=""
exec=""
action=""
verbose="false"
no_default_libs="false"

gcc=${GCC-gcc}
objdump=${OBJDUMP-objdump}
objcopy=${OBJCOPY-objcopy}

# =================
# GRAPH ABSTRACTION
# =================
#
# Because we do some hairy graph operations, we provide some
# abstractions of them. Some functions here are very simple, but
# the source is much more readable this way.
```

```
# check_node NODE ...
#   checks if all NODEs are valid node names.
# Used internally for verificaton only.
# Return 0 if all NODEs are valid.
# Currently, a node is valid if it does not contain a space.

check-node () {
  local node
  for node in "$@" ; do
    if [ "x`echo $node | sed -e '/ /d'`" = x ] ; then
      echo 1>&2 $0: check-node: invalid node \"$node\"
      exit 1
    fi
  done
  return 0
}

# is-graph FILE ...
#   provides a very simple type assertion
# Turns FILE into a graph if it isn't already and returns 0.

is-graph () {
  local file
  for file in "$@" ; do
    if [ ! -e "$file" ] ; then
      touch "$qfile"
    fi
  done
}

# add-node FILE NODE
#   add a node NODE to graph FILE.
# This is useful if you need to make sure that a node appears
# in the graph without actually connecting it to an arrow.
# You don't need to add nodes that are part of an arrow.

add-node () {
  if [ $# != 2 ] ; then
    echo 1>&2 $0: add-node: internal error: called with invalid number of arguments
    exit 1
  fi
  check-node "$2"
  echo "$2 $2" >> "$1"
  return 0
}

# add-arrow FILE NODE1 NODE2
#   add an arrow from NODE1 to NODE2 to graph FILE.
```

```
add-arrow () {
  if [ $# != 3 ] ; then
    echo 1>&2 $0: add-arrow: internal error: called with invalid number of arguments
    exit 1
  fi
  check-node "$2" "$3"
  echo "$2 $3" >> "$1"
  return 0
}

# find-cycle FILE
#   finds a cycle in a graph FILE.
# If a cycle is found, it is printed out at stdin, one node each line,
# and 0 is returned. Otherwise, nothing is printed  on stdout and exit
# status is 1.

find-cycle () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: find-cycle: internal error: called with invalid number of arguments
    exit 1
  fi
  tsort "$1" 2> "$f1_dir/find-cycle" > /dev/null
  if [ ! -s $f1_dir/find-cycle ] ; then
    return 1
  else
    if [ "x`head -1 $f1_dir/find-cycle`" != "xtsort: cycle in data" ] ; then
      echo 1>&2 $0: find-cycle: internal error: tsort has invalid output format
      return 2
    fi
    cat "$f1_dir/find-cycle" | sed -e '1d' -e '/tsort: cycle in data/,$d' -e 's/^tsort:
//'
  fi
}

# shrink-nodes FILE NODE1 ...
#   shrinks several nodes NODE1 ... to a single node in graph FILE.
# To hide cycles, we treat a cycle as a single node and replace
# each occurence of a node in the cycle with a new node
# [NODE1,...] . This change is destructive and can not be undone!
# (You would need to store the entry point to the cycle for each arrow
# pointing to/from it).
# This function does not check if the the nodes NODE1 ... exist.
# However, if none of these nodes exists already, the new node will
# not appear either. This makes this function sort of idem potent.
# It does not check if NODE1 ... are a cycle. We will assume this
# later in the library dependency analysis, but nothing in the code
# relies on it.
```

```
# Always shrink all cycles, or you may get unresolved symbols.
#
# Example:
#  N1  --->  N2                         N1 -------> /------------\
#  |             "shrink-nodes N2 N4"   |         _  | [N2,N4]    |
#  v             ------------------->   v    _____/| \------------/
#  N3  --->  N4                         N3 /

# A small helper function will aid us...
# equal-match STRING STRING1 ...
#   return 0 if STRING is among STRING1 ..., 1 otherwise.
equal-match () {
  local string
  local stringk
  string="$1"
  shift
  for stringk in "$@" ; do
    if [ "x$string" = "x$stringk" ] ; then
      return 0
    fi
  done
  return 1
}

shrink-nodes () {
  local head
  local lnode
  local rnode
  local graph="$1"
  shift
  is-graph "$graph"
  check-node "$@"
  local cnode="[`echo "$@" | sed 's/ /,/g'`]"
  # Okay, it's a hack. We treat the graph as a queue. I am just too
  # lazy to copy the relevant code here. Of course, we exploit several
  # properties of the graph and queue file format here (for example,
  # that graphs never can contain a QUEUE_SEPERATOR, and that a graph is
  # really a simple file with "a b" entries).
  cat /dev/null > "$fl_dir/shrink-cycle"
  while head=`get-top-of-queue "$graph"` ; do
    lnode=`echo $head|sed 's/ [^ ]*$//'`
    if equal-match "$lnode" "$@" ; then
      lnode="$cnode"
    fi
    rnode=`echo $head|sed 's/^[^ ]* //'`
    if equal-match "$rnode" "$@" ; then
      rnode="$cnode"
    fi
```

```
    echo "$1node $rnode" >> "$f1_dir/shrink-cycle"
  done
  cat "$f1_dir/shrink-cycle" | sort -u > "$graph"
}

# =================
# QUEUE ABSTRACTION
# =================
#
# I added an abstract interface for queues to make the code more readable.
# Queue operations usually consist of several atomic file operations, which
# can get quite messy.
#
# You can use queues to simply loop through all lines of a file, but you
# also can add stuff to the queue while processing it.
#
# Implementation: All queues consist of a QUEUE_FILE which has two parts:
# the remaining entries in the queue (QUEUE) and the already processed
# entries (BUCKET).
# The two parts are seperated by a line containing only QUEUE_SEPERATOR.

QUEUE_SEPERATOR=SEPERATOR___ABOVE_IS_QUEUE__BELOW_IS_BUCKET___SEPERATOR

# check-queue-entry QENTRY ...
#   checks if all queue entries QENTRY are valid.
# Used internally for verificaton only.
# Return 0 if all QENTRYs are valid.
# Currently, a node is valid if it does not match the QUEUE_SEPERATOR.

check-queue-entry () {
  local qentry
  for qentry in "$@" ; do
    if [ "x`echo $qentry | sed "/^$QUEUE_SEPERATOR$/d"`" = x ] ; then
      echo 1>&2 $0: check-queue-entry: invalid qentry name \"$qentry\"
      exit 1
    fi
  done
  return 0
}

# is-queue QUEUE_FILE ...
#   provides a very simple type assertion
# Turns QUEUE_FILE into a queue if it isn't already and returns 0.

is-queue () {
  local qfile
  for qfile in "$@" ; do
    if [ ! -e "$qfile" ] ; then
```

```
      echo "$QUEUE_SEPERATOR" > "$qfile"
    else
      if ! grep -q "^$QUEUE_SEPERATOR$" "$qfile" ; then
        echo "$QUEUE_SEPERATOR" >> "$qfile";
      fi
    fi
  done
}

# get-top-of-queue QUEUE_FILE
#   processes a queue one more time.
# If QUEUE of QUEUE_FILE is empty, exit status is 1 and no output is given.
# Otherwise, top of QUEUE is removed, returned on stdout and
# appended to the end of the BUCKET part of QUEUE_FILE.

get-top-of-queue () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: get-top-of-queue: internal error: called with invalid number of argum
ents
    exit 1
  fi
  is-queue "$1"
  local head=`head -1 "$1"`
  if [ "x$head" = "x$QUEUE_SEPERATOR" ] ; then
    return 1
  else
    sed -e 1d "$1" > "$f1_dir/get-top-of-queue"
    echo "$head" | tee --append "$f1_dir/get-top-of-queue"
    cat "$f1_dir/get-top-of-queue" > "$1"
    return 0
  fi
}

# add-to-queue-if-not-there QUEUE_FILE QENTRY ...
#   add queue entries QENTRY ... to the beginning of the
# QUEUE of QUEUE_FILE if it is neither in QUEUE nor in BUCKET
# of QUEUE_FILE.
# Return with exit status 0.
# Note: If you want to add QENTRY to the *end* of QUEUE, you would do
# something like the following:
# sed -e s/^$QUEUE_SEPERATOR$/$head"'\
# '"$QUEUE_SEPERATOR/"
# which is necessary to pass the newline to sed. I think we can take the
# easy way out.

add-to-queue-if-not-there () {
  local qentry
  local qfile="$1"
```

```
  shift
  check-queue-entry "$@"
  is-queue "$qfile"
  for qentry in "$@" ; do
    if ! grep -q "^$qentry\$" "$qfile" ; then
      echo "$qentry" > "$fl_dir/add-to-queue-if-not-there"
      cat "$qfile" >> "$fl_dir/add-to-queue-if-not-there"
      cat "$fl_dir/add-to-queue-if-not-there" > "$qfile"
    fi
  done
  return 0
}

pop-to-top-of-queue () {
  local qfile="$1"
  local qentry="$2"             # can be a prefix
  local fullentry
  check-queue-entry "$2"
  is-queue "$qfile"
  if grep -q "^$qentry" "$qfile" ; then
    # get the full name
    fullentry=`grep "^$qentry" "$qfile"`
    # this entry is in the queue, so first we remove it
    sed -e "/^$qentry/d" $qfile > "$fl_dir/pop-to-top-of-queue"
    # lets make sure there's not another entry which matches
    if grep -q "^$qentry" "$fl_dir/pop-to-top-of-queue" ; then
      echo 1>&2 $0: pop-to-top-of-queue: two entries match $2
      exit 1
    fi
    cat "$fl_dir/pop-to-top-of-queue" > "$qfile"
    # now add it back; it will go on the top
    add-to-queue-if-not-there "$qfile" "$fullentry"
  fi
}

# ==================
# LIBRARY PROCESSING
# ==================
#
# The following helper functions mess around with the actual
# processing and installation of libraries.
#

# get-library-depends OBJ1 ...
#   get all libraries the objects OBJ1 ... depend on.
# OBJs can be binaries or shared libraries.
# The list is neither sort'ed nor uniq'ed.
```

```
get-library-depends () {
  if [ $# = 0 ] ; then
    echo 1>&2 $0: get-library-depends: internal error: no arguments
    exit 1
  fi
  $objdump --private-headers "$@" 2> /dev/null \
    | sed -n 's/^ *NEEDED *\([^ ]*\)$/\1/p'
}

# get-undefined-symbols OBJ1 ...
#   get all unresolved symbols in OBJ1 ...
# The list is neither sort'ed nor uniq'ed.

get-undefined-symbols () {
  if [ $# = 0 ] ; then
    echo 1>&2 $0: get-undefined-symbols: internal error: no arguments
    exit 1
  fi
  # ash has undefined reference to sys_siglist if .bss is not mentioned
  # here. Reported by Joel Klecker.
  # All symbols are epxosed, so we just catch all. Suggested by Roland
  # McGrath. Another thing to try is to investigate --dynamic-reloc.
  $objdump --dynamic-syms "$@" 2> /dev/null \
    | sed -n 's/^.* \([^ ]*\)$/\1/p'
#   | sed -n 's/^.*[\*UND\*|.bss].* \([^ ]*\)$/\1/p'
}

# get-provided-symbols LIB1 LIB2 ...
#   get all symbols available from libraries LIB1 ... .
# Does only work for pic libraries.
#
#                       v Watch the tab stop here.
# 00000000  w    F .text  00000000              syscall_device_write_request
# 00000000 g     F .text  0000056c              __strtoq_internal

get-provided-symbols () {
  if [ $# = 0 ] ; then
    echo 1>&2 $0: get-provided-symbols: internal error: no arguments
    exit 1
  fi
  $objdump --syms "$@" 2>/dev/null | grep -v '\*UND\*' \
    | sed -n 's/^[0-9a-f]\+ \(g \| w\)   .. .*      [0-9a-f]\+ \(0x8[08]\)\? *\([^ ]*\)$
/\3/p'
}

# Crude hack (?) only used for diagnostic.

get-provided-symbols-of-so-lib () {
```

```
  if [ $# = 0 ] ; then
    echo 1>&2 $0: get-provided-symbols: internal error: no arguments
    exit 1
  fi
  $objdump --dynamic-syms "$@" 2>/dev/null \
    | sed -e '/\*UND\*/d' | sed -n 's/^.* \([^ ]*\)$/\1/p'
}

# get-common-symbols FILE1 FILE2
#   returns a list of all symbols in FILE1 that appear also in FILE2
# Note: When get-common-symbols returns, FILE1 and FILE2 are "sort -u"'ed.
# Note: Version Information in FILE1 is ignored when comparing.

get-common-symbols () {
  if [ $# != 2 ] ; then
    echo 1>&2 $0: get-common-symbols: internal error: called with invalid number of arg
uments
    exit 1
  fi
  # Not needed anymore, but we go for compatibility.
  # (Somewhere we HAVE to clean FILE2 up).
  sort -u "$1" > $f1_dir/get-common-symbols
  cat $f1_dir/get-common-symbols > "$1"
  sort -u "$2" > $f1_dir/get-common-symbols
  cat $f1_dir/get-common-symbols > "$2"

  local symbol=
  while symbol=`get-top-of-queue $f1_dir/get-common-symbols` ; do
    grep ^$symbol\$\\\|^$symbol@ "$1"
  done
}

# create-link TARGET LINK_NAME
#  creates a soft link if there isn't one already.

create-link () {
  if [ $# != 2 ] ; then
    echo 1>&2 $0: create-link: internal error: called with invalid number of arguments
    exit 1
  fi
  if [ ! -e "$2" ] ; then
    $action ln -sf "$1" "$2"
  fi
}



# find-file-glob PATH FILE
#   search all directories in PATH for file FILE, return absolute path
```

```
# FILE can be a regular expression, and/or a relative path to a file.
# PATH is a list, seperator is ':'.

find-file-glob () {
  if [ $# != 2 ] ; then
    echo 1>&2 $0: find-file-glob: internal error: exactly two arguments required
    exit 1
  fi
  local path=$1
  export dir=`echo $path | sed -e 's/:.*$//'`
  export regex=$2
  until [ "x$path" = x ] ; do
    if [ "x$dir" != x ] ; then
      file=`/bin/bash -c 'ls $dir/$regex' 2> /dev/null`
      if [ "$file" ] ; then
        second_match=`echo $file | cut -s -d " " -f 2`
       if [ $second_match ] ; then
          echo 1>&2 $0: find-file-glob: $dir/$regex internal error: multiple matches
        exit 1
       fi
        echo "$file"
        return 0
      fi
    fi
    path=`echo $path | sed -e 's/^[^:]*:*//'`
    dir=`echo $path | sed -e 's/:.*$//'`
  done
  return 1
}


# find-file PATH FILE
#   search all directories in PATH for file FILE, return absolute path
# FILE can be a relative path and a filename.
# PATH is a list, seperator is ':'.

find-file () {
  if [ $# != 2 ] ; then
    echo 1>&2 $0: find-file: internal error: exactly two arguments required
    exit 1
  fi
  local path=$1
  local dir=`echo $path | sed -e 's/:.*$//'`
  until [ "x$path" = x ] ; do
    if [ "x$dir" != x ] ; then
      if [ -e "$dir/$2" ] ; then
        echo "$dir/$2"
        return 0
```

```
      fi
    fi
    path=`echo $path | sed -e 's/^[^:]*:*//'`
    dir=`echo $path | sed -e 's/:.*$//'`
  done
  return 1
}

# find-files PATH FILE1 FILE2 ...
#   search all directories in PATH for file FILE1, FILE2...
# FILE can be a relative path and a filename.
# PATH is a list, seperator is ':'.
# Return value is a white space seperated list of absolute filenames.

find-files () {
  if [ $# -lt 2 ] ; then
    echo 1>&2 $0: find-files: internal error: too few arguments
    exit 1
  fi
  local path="$1" ; shift
  while [ $# != 0 ] ; do
    find-file $path $1
    shift
  done
}

# get-pic-file LIB
#   returns the filename of the pic archive for LIB.
# Note: There doesn't seem to be any convention, *ick*.

get-pic-file () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: get-pic-file: internal error: called with invalid number of arguments
    exit 1
  fi
  case "$1" in
    libc-*.so)
      # For libc.so, we have some extra files to include. The older 2.1.x
      # lib uses .so for the object names, while newer ones use .o. Also,
      # some of them have an interp object. This should work in all cases.
      echo `find-files $src_path libc_pic/soinit.so libc_pic/soinit.o \
            libc_pic.a libc_pic/sofini.so libc_pic/sofini.o \
            libc_pic/interp.o`
      ;;
    *)
      local libname=`echo $1 | sed -e 's/^lib\([^-.]*\)[-.].*/\1/'`
      echo `find-file-glob $src_path lib${libname}*_pic.a`
      ;;
```

```
  esac
  return 0
}

get-extra-flags () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: get-extra-flags: internal error: called with invalid number of argume
nts
    exit 1
  fi
  flags=""

  case "$1" in
    libc-[0-9]*.so)
      # libc.so is a special case, we need the dynamic linker aswell
      local ver=`echo $1 | sed -e 's/^libc-\(.*\).so/\1/'`
      flags="`find-file $src_path ld-${ver}.so`"
      ;;
  esac
  local libname=`echo $1 | sed -e 's/^lib\([^-.]*\)[-.].*/\1/'`
  local map=`find-file $src_path lib${libname}_pic.map`
  test -z "$map" || \
    flags="$flags -Wl,--version-script=${map}"
  echo $flags
  return 0
}

# install-small-lib LIB_SONAME
#   makes a small version of library LIB_SONAME
#
# This happens the following way:
# 0. Make exception for the linker ld.
# 1. Try to figure out complete path of pic library.
# 2. If no found, copy the shared library, else:
#  a. Get shared libraries this lib depends on, transform into a
#     list of "-lfoo" options.
#  b. Get a list of symbols both provided by the lib and in the undefined
#     symbols list.
#  c. Make the library, strip it.
#  d. Add symbols that are still undefined to the undefined symbols list.
#  e. Put library into place.

install-small-lib () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: install-small-lib: internal error: called with invalid number of argu
ments
    exit 1
  fi
```

```
  local src_file=`find-file $src_path $1`
  if `echo "$1" | grep -q ^ld` ; then
    get-provided-symbols "$src_file" >> $fl_dir/provided-symbols
    $action $objcopy --strip-unneeded -R .note -R .comment "$src_file" "$dest/$1"
    chmod 755 "$dest/$1"
    return 0
  fi
  local pic_objects=`get-pic-file "$1"`
  local extra_flags=`get-extra-flags "$1"`
  if [ "x$pic_objects" = x ] ; then
    $verbose 2>&1 No pic archive for library "$1" found, falling back to simple copy.
    get-provided-symbols-of-so-lib "$src_file" >> $fl_dir/provided-symbols
    get-undefined-symbols "$src_file" >> $fl_dir/undefined-symbols
    $action $objcopy --strip-unneeded -R .note -R .comment "$src_file" "$dest/$1"
  else
    $verbose 2>&1 Make small lib from "$pic_objects" in "$dest/$1".

    # XXX: If ld is NEEDED, we need to include it on the gcc command line
    get-library-depends "$src_file" \
      | sed -n -e 's/^lib\([^-.]*\)[-.].*/\1/p' > $fl_dir/lib-dependencies
    get-provided-symbols $pic_objects > $fl_dir/lib-provided-symbols
    # Argument order does matter:
    get-common-symbols $fl_dir/lib-provided-symbols \
      $fl_dir/undefined-symbols > $fl_dir/lib-symbols-to-include

    local soname=`objdump -x $src_file 2>&1 | grep SONAME | awk '{print $2}'`

    ($verbose && set -x; ${gcc} \
      -nostdlib -nostartfiles -shared \
      "-Wl,-soname=$soname" \
      `cat $fl_dir/lib-symbols-to-include | sed 's/^/-u/'` \
      -o $fl_dir/lib-so $pic_objects $extra_flags -lgcc "-L$dest" \
      -L`echo $src_path | sed -e 's/::*/:/g' -e 's/^://' -e 's/:$//' \
      -e 's/:/ -L/g'` \
      `cat $fl_dir/lib-dependencies | sed 's/^/-l/'`) \
     && $objcopy --strip-unneeded -R .note -R .comment $fl_dir/lib-so $fl_dir/lib-so-s
tripped \
     || {
       echo 1>&2 $0: install-small-lib: $gcc or $objcopy failed.
       exit 1
     }
    get-undefined-symbols $fl_dir/lib-so-stripped \
      >> $fl_dir/undefined-symbols
    get-provided-symbols-of-so-lib $fl_dir/lib-so-stripped >> $fl_dir/provided-symbols
    $action cp $fl_dir/lib-so-stripped "$dest/$1"
  fi
}
```

```
# install-libs-in-sphere [LIB1,...]
#   extracts the libs in a shrinked node and cycles through them until all
# possible symbols are resolved.
# Always make sure this can be called recursively (from install-libs)!

install-libs-in-sphere () {
  if [ $# != 1 ] ; then
    echo 1>&2 $0: install-libs-in-sphere: internal error: called with invalid number of
arguments
    exit 1
  fi
  # Unfortunately, we need a small parser here to do the right thing when
  # spheres are within spheres etc. RegEx simply can't count brackets. :(
  local string=`echo "$1" | sed -e 's/^\[//' -e 's/\]$//'`
  local char
  local result=
  local depth=0
  while [ "x$string" != x ] ; do
    # Jump to next special char for faster operation.
    # Don't be confused by the regex, it matches everything but ],[
    char=`echo $string | sed -e 's/^\([^],[]*\).*$/\1/'`
    string=`echo $string | sed -e 's/^[^],[]*//'`
    result="$result$char";
    # Read special char
    char=`echo $string | sed -e 's/^\(.\).*$/\1/'`
    string=`echo $string | sed -e 's/^.//'`
    case "$char" in
      [) depth=$(($depth+1));;
      ]) depth=$(($depth-1));;
      ,) if [ $depth = 0 ] ; then
           char=' ';
         fi;;
    esac
    result="$result$char";
  done
  $verbose 2>&1 "RESOLVING LOOP...`echo $result | md5sum`"
  echo XXX: CODE NOT FINISHED
  install-libs $result
  $verbose 2>&1 "END OF LOOP... `echo $result | md5sum`"
}

# install-libs LIB1 ...
#   goes through an ordered list of libraries and installs them.
# Make sure this can be called recursively, or hell breaks loose.
# Note that the code is (almost) tail-recursive. I wish I could
# write this in Scheme ;)

install-libs () {
```

```
  local cur_lib
  local lib
  for cur_lib in "$@" ; do
    if echo "$cur_lib" | grep -q '^\[' ; then
      install-libs-in-sphere "$cur_lib"
    else
      lib=`find-file $src_path $cur_lib`
      if [ -L "$lib" ] ; then
        lib=`basename \`readlink $lib\``
        create-link $lib $dest/$cur_lib
      else
        install-small-lib $cur_lib
      fi
    fi
  done
}

#
#  MAIN PROGRAM
#
#  1. Option Processing
#  2. Data Initialization
#  3. Graph Construction and Reduction
#  4. Library Installation

#  Global Files:
#  $fl_dir/undefined-symbols
#     Holds all undefined symbols we consider for inclusion.
#     Only grows. Does not to be sort'ed and uniq'ed, but will
#     get occasionally.
#  $fl_dir/provided-symbols
#     Holds all defined symbols we included.
#     Only grows. Should later be a superset of undefined-symbols.
#     But some weak symbols may be missing!
#  $fl_dir/library-depends
#     Queue of all libraries to consider.

#
# 1. Option Processing
#

while :; do
  case "$1" in
    -L) src_path="$src_path:$2"; shift 2;;
    -d|--dest-dir) dest=$2; shift 2;;
    -n|--dry-run) action="echo"; shift;;
    -v|--verbose) verbose="echo"; shift;;
    -V|--version) echo "$version"; exit 1;;
```

```
    -D|--no-default-lib) no_default_libs="true"; shift;;
    -h|--help)
      echo "$usage"
      echo "Make a set of minimal libraries for FILE ... in directory DEST."
      echo ''
      echo "\
Options:
  -L DIRECTORY               Add DIRECTORY to library search path.
  -D, --no-default-lib       Do not use default lib directories of $default_src_path
  -n, --dry-run              Don't actually run any commands; just print them.
  -v, --verbose              Print additional progress information.
  -V, --version              Print the version number and exit.
  -h, --help                 Print this help and exit.

  -d, --dest-dir DIRECTORY   Create libraries in DIRECTORY.

Required arguments for long options are also mandatory for the short options."
      exit 0;;
    -*) echo 1>&2 $0: $1: unknown flag; echo 1>&2 "$usage"; echo 1>&2 "$try"; exit 1;;
    ?*) exec="$exec $1"; shift;;
    *)  break;;
  esac
done

if $no_default_libs; then
       src_path=${src_path:-$default_src_path}
else
       src_path=${src_path-$default_src_path}
fi

src_path=`echo $src_path | sed 's,^:*,,'`

if [ "x$exec" = x ] ; then
  exit 0
fi
if [ "x$dest" = x ] ; then
  echo 1>&2 $0: no destination directory given; echo 1>&2 "$usage"; exit 1
fi



#
# 2. Data Initialization
#

$verbose -n 2>&1 "Initializing data objects... "

# Temporary directory.
fl_dir="/tmp/,mklibs.$$"
```

```
set -e
mkdir $fl_dir
set +e

trap "rm -fr $fl_dir" EXIT

# Intialize our symbol array and library queue with the information
# from the executables.

get-undefined-symbols $exec > $fl_dir/undefined-symbols
add-to-queue-if-not-there $fl_dir/library-depends `get-library-depends $exec`

# reorder libraries a bit

pop-to-top-of-queue "$fl_dir/library-depends" libslang.so
pop-to-top-of-queue "$fl_dir/library-depends" libnewt.so

$verbose 2>&1 "done."


#
# 3.a Graph Construction
#
# Build the dependency graph, add new library dependencies to the queue on
# the way.
#   If the soname is a link, add the target to the end of the queue and
#   add a simple arrow to the graph.
#   If the soname is a real lib, get its dependencies and add them to
#   the queue. Furthermore, add arrows to the graph. If the lib is not
#   dependant on any other lib, add the node to make sure it is mentioned
#   at least once in the graph.

$verbose -n 2>&1 "Constructing dependency graph... ("

while cur_lib=`get-top-of-queue $fl_dir/library-depends`
do
  lib=`find-file $src_path $cur_lib`
  if [ -L "$lib" ] ; then
    $verbose -n 2>&1 L
    lib=`basename \`readlink $lib\``
    add-to-queue-if-not-there $fl_dir/library-depends "$lib"
    add-arrow $fl_dir/dependency-graph "$cur_lib" "$lib"
  else
    get-library-depends "$lib" > $fl_dir/backup
    if [ "x`head -1 $fl_dir/backup`" = x ] ; then
      $verbose -n 2>&1 N
      add-node $fl_dir/dependency-graph "$cur_lib"
    else
```

```
      $verbose -n 2>&1 A
      for lib in `cat $f1_dir/backup` ; do
        add-to-queue-if-not-there $f1_dir/library-depends "$lib"
        add-arrow $f1_dir/dependency-graph "$cur_lib" "$lib"
      done
    fi
  fi
done

$verbose 2>&1 ") done."

if [ ! -f $f1_dir/dependency-graph ]; then
  echo 1>&2 mklibs: internal error: $f1_dir/dependency-graph not generated
  exit 1
fi

#
# 3.b Graph Reduction
#
# Find and shrink cycles in the graph.

$verbose -n 2>&1 "Eliminating cycles... ("

while cycle=`find-cycle "$f1_dir/dependency-graph"` ; do
  $verbose -n 2>&1 C
  shrink-nodes "$f1_dir/dependency-graph" $cycle
done

$verbose 2>&1 ") done."

#
# 4. Library Installation
#
# Let tsort(1) do the actual work on the cycle-free graph.

tsort $f1_dir/dependency-graph > $f1_dir/backup

# Now the ordered list of libraries (or cycles of them)
# can be processed by install-libs. This is indeed the last step.

install-libs `cat $f1_dir/backup`

#sort -u $f1_dir/provided-symbols > $f1_dir/diag1
#sort -u $f1_dir/undefined-symbols > $f1_dir/diag2
#cat $f1_dir/diag1 $f1_dir/diag2 | sort | uniq -u > $f1_dir/diag3
## diag3 has now the symmetric difference.
#cat $f1_dir/diag3 $f1_dir/diag2 | sort | uniq -d > $f1_dir/diag1
## diag1 has now all undefined symbols that are not provided.
```

```
##cat $f1_dir/diag1 | wc
## Note that some of these symbols are weak and not having them is probably
## not an error.

exit 0
```