Linux Library Handling

The possible candidates, in order of age, are:

- libc 4, a.out: very old systems
- libc 5, ELF: Red Hat 4.2, Debian 2.0
- libc 6 (a.k.a glibc 2), ELF: Red Hat 5 - 5.2, Debian 2.1
- libc 6.1,(a.k.a glibc 2.1) ELF: Red Hat 6

C-library and header files

binutils:

ar- create, modify, and extract from archives
as- the portable GNU assembler
gprof - display call graph profile data
ld- the GNU linker
nm - list symbols from object files
objcopy - copy and translate object files
objdump - display information from object files
ranlib - generate index to archive
readelf - Displays information about ELF files
size - list section sizes and total size
strings - print the strings of printable characters in files
strip - Discard symbols from object files

How to compile programs with Autotools

Programs that use Autotools use these three commands:

```
# ./configure
# make
# make install
```

Sometimes you also add options to these three commands, but that's usually all you do to compile and install the program.

- Here, ./configure analyzes your system to see what kind of programs and libraries you have, so it knows how to build the program best.

- Then make does the actual building.

- Lastly, make install installs the program.

Autotools is a collection of three tools:

* Autoconf - This is used to generate the "configure" shell script. This is the script that analyzes your system at compile-time. For example, does your system use "cc" or "gcc" as the C compiler?

* Automake - This is used to generate Makefiles. It uses information provided by Autoconf. For example, if your system has "gcc", it will use "gcc" in the Makefile. Or, if it finds "cc" instead, will use "cc" in the Makefile.

* Libtool - This is used to create shared libraries, platform-independently.

This semester we are skipping the rest of the material. If you plan to distribute a package, you can go through the rest of the material. It is not part of the course !!!

I am keeping the material for reference purpose.

```
/* hello.c: A standard "Hello, world!" program */

#include <stdio.h>

int main(int argc, char* argv[])
{
   printf("Hello, world!\n");

   return 0;
}
```

Listing of "Makefile"

```
# Makefile: A standard Makefile for hello.c

all: hello

clean:
	rm -f hello *.o
```

```
$ ls
Makefile  hello.c
$ make
cc     hello.c   -o hello
$ ls
Makefile  hello*  hello.c
$ ./hello
Hello, world!
```

Let's add Autoconf

First, we create a file called "configure.ac". This file instructs Autoconf how to generate the "configure" script.

Creating this file by hand can be tedious, though. So Autoconf provides a program "autoscan":

```
$ autoscan
$ ls
Makefile  autoscan.log  configure.scan  hello*  hello.c
```

"autoscan" scans sources and creates an appropriate "configure.ac" file. But notice it doesn't create "configure.ac" file directly. Instead, it creates a file called "configure.scan". We need to rename this file to "configure.ac":

```
$ mv configure.scan configure.ac
```

Anyway, now we can run "autoconf" to generate "configure":

```
$ autoconf
$ ls
Makefile         autoscan.log  configure.ac  hello.c
autom4te.cache/  configure*    hello*
$
```

- "configure" is supposed to generate a new Makefile.
- But "configure" uses a file called "Makefile.in" to generate the new Makefile.
- "Makefile.in" is supposed to be a template for the new "Makefile".
- We'll simply rename "Makefile" to "Makefile.in" and worry about writing the proper version later.

```
$ mv Makefile Makefile.in
$ ls
Makefile.in      autoscan.log  configure.ac  hello.c
autom4te.cache/  configure*    hello*
$
```

Now we're ready to run "./configure" and "make":

```
$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
.....................................
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
config.status: error: cannot find input file: config.h.in
```

```
$ make clean all
rm -f hello
cc     hello.c   -o hello
$ ls
Makefile     autom4te.cache/  config.log     configure*    hello*
Makefile.in  autoscan.log     config.status*  configure.ac
hello.c
$ ./hello
Hello, world!
$
```

Let's review what we've done so far:

```
# Create the sources, Makefile, etc.
# Run "autoscan"
  ("autoscan" creates "configure.scan")
# Rename "configure.scan" to "configure.ac"
# Run "autoconf"
  ("autoconf" uses "configure.ac" to create "configure")
# Rename "Makefile" to "Makefile.in"
  ("configure" will use "Makefile.in" to create "Makefile")
# Run "./configure" and "make"
```

We added Autoconf to our program. But it's not quite portable yet. So we'll try to make it portable. "config.h" has portability constants.

To make the program portable, we'll need to modify our sourcefile, "hello.c". Our sourcefile will need #ifdef, #ifndef, etc. to be portable.

But we can't use #ifdef and #ifndef unless we have some constants to check.

Well, it turns out the constants we need are in "config.h".

So we'll simply #include "config.h".

•But we don't have "config.h" yet.
•"config.h" is created by Autoconf, and we need to tell Autoconf to generate one.
•run "autoheader", followed by "./configure"

```
$ ls
Makefile     autom4te.cache/  config.log      configure*    hello*
Makefile.in  autoscan.log     config.status*  configure.ac  hello.c
$ autoheader
autoheader2.50: `config.h.in' is created
$ ls
Makefile         autoscan.log  config.status*  hello*
Makefile.in      config.h.in   configure*      hello.c
autom4te.cache/  config.log    configure.ac
```

```
$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
$ ls
Makefile         autoscan.log  config.log      configure.ac
Makefile.in      config.h      config.status*  hello*
autom4te.cache/  config.h.in   configure*      hello.c
$
```

Notice how "autoheader" generates "config.h.in", then "config.h.in" was used by "configure" to generate "config.h"?

In general, that's how Autoconf works:

We use some program to generate "<something>.in", and "<something>.in" will be used by Autoconf to generate "<something>".

Listing of "config.h":

```
/* config.h.  Generated by configure.  */
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Define to the address where bug reports for this package
should be sent. */
#define PACKAGE_BUGREPORT "BUG-REPORT-ADDRESS"

/* Define to the full name of this package. */
#define PACKAGE_NAME "FULL-PACKAGE-NAME"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "FULL-PACKAGE-NAME VERSION"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "full-package-name"

/* Define to the version of this package. */
#define PACKAGE_VERSION "VERSION"
```

•config.h : There are some constants, but nothing useful for portability.

•Our program is too simple to have any portability issues.

•We need to make our program more complex, so we actually get something in "config.h".

•Add complexity to "hello.c"

•Modify our program to print the number of seconds since the Epoch (January 1, 1970 00:00:00 GMT) using the gettimeofday function call

```
/* hello.c: A program to show the time since the Epoch */

#include <stdio.h>
#include <sys/time.h>


int main(int argc, char* argv[])
{
   double sec;
   struct timeval tv;

   gettimeofday(&tv, NULL);
   sec = tv.tv_sec;
   sec += tv.tv_usec / 1000000.0;

   printf("%f\n", sec);

   return 0;
}
```

```
$ autoscan
Use of uninitialized value in concatenation (.) or string at
/usr/bin/autoscan line 195.
configure.ac: warning: missing
AC_CHECK_FUNCS([gettimeofday]) wanted by: hello.c :12
configure.ac: warning: missing
AC_CHECK_HEADERS([sys/time.h]) wanted by: hello.c :4
configure.ac: warning: missing AC_HEADER_TIME wanted by:
hello.c:10
$ mv configure.scan configure.ac
$ autoconf
$ autoheader
autoheader2.50: `config.h.in' is updated
```

```
$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
.....................................
checking for stdint.h... yes
checking for unistd.h... yes
checking sys/time.h usability... yes
checking sys/time.h presence... yes
checking for sys/time.h... yes
checking whether time.h and sys/time.h may both be included...
yes
checking for gettimeofday... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
```

Notice a few things here:

• After running "autoscan", remember to rename "configure.scan" to "configure.ac".

• "autoscan" generates warning messages about "configure.ac".

• This is because "configure.ac" is checked by "autoscan" to see if it needs updating.

• In our case, we simply replace "configure.ac" with "configure.scan".

• We may not want to do this in the future, if we have a customized version of "configure.ac". If that were the case, we would edit "configure.ac" manually instead.

```
/* config.h.  Generated by configure.  */
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Define to 1 if you have the `gettimeofday' function. */
#define HAVE_GETTIMEOFDAY 1

/* Define to 1 if you have the <inttypes.h> header file. */
#define HAVE_INTTYPES_H 1

/* Define to 1 if you have the <memory.h> header file. */
#define HAVE_MEMORY_H 1

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1
.......................
```

```
/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/time.h> header file. */
#define HAVE_SYS_TIME_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* Define to the address where bug reports for this package
should be sent. */
#define PACKAGE_BUGREPORT "BUG-REPORT-ADDRESS"
```

```
/* Define to the full name of this package. */
#define PACKAGE_NAME "FULL-PACKAGE-NAME"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "FULL-PACKAGE-NAME VERSION"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "full-package-name"

/* Define to the version of this package. */
#define PACKAGE_VERSION "VERSION"

/* Define to 1 if you have the ANSI C header files. */
#define STDC_HEADERS 1

/* Define to 1 if you can safely include both <sys/time.h> and
<time.h>. */
#define TIME_WITH_SYS_TIME 1
```

Using this header file, we can check for several things.

For simplicity, though, we'll check only two things --

- Whether gettimeofday() exists (using "HAVE_GETTIMEOFDAY"), and

- whether we have "sys/time.h" (using "HAVE_SYS_TIME_H").

```
/* hello.c: A program to show the time since the Epoch */

#include <stdio.h>
#include "config.h"

#ifdef HAVE_SYS_TIME_H
#include <sys/time.h>
#else
#include <time.h>
#endif

double get_sec_since_epoch()
{
  double sec;

  #ifdef HAVE_GETTIMEOFDAY
    struct timeval tv;

    gettimeofday(&tv, NULL);
    sec = tv.tv_sec;
    sec += tv.tv_usec / 1000000.0;
  #else
    sec = time(NULL);
  #endif

  return sec;
}

int main(int argc, char* argv[])
{
  printf("%f\n", get_sec_since_epoch());   return 0;}
```

A separate function to handle the time function call.

This way one can group the functions that have portability problems into one section of the code, and adjust it again in the future if necessary.

The time function call tries to use the gettimeofday() system call, but it will fall back on the time() system call if gettimeofday() isn't available.

gettimeofday() is more useful because it also provides microsecond information, but we'll use time() if gettimeofday() isn't available.

Don't forget to #include "config.h".

Write your program, keeping portability in mind. Create "Makefile.in".

    # Run "autoscan".

    # Rename "configure.scan" to "configure.ac"

    # Run "autoheader".

    # Run "autoconf".

    # "./configure".

    # Check "config.h". If necessary, modify source and repeat from step #2.

    # And lastly, compile!

What is Automake?

Automake is a program that will automatically generate the "Makefile" for you.

Although we already have a working "Makefile", it's not quite portable.

For example, should the Makefile use "cc" or "gcc"? Or should the name of the executable be "myprog" or "myprog.exe"?

Portability doesn't only affect our sourcecode, but our Makefile as well.

Automaking involves three steps. But you should already be familiar with step #3:

```
    # Create a file named "Makefile.am".
    # "Makefile.am" will be used by Automake to create a new "Makefile.in".
    # "Makefile.in" will be used by "configure" to create "Makefile".
```

We have already done #3 before. So we just need to create "Makefile.am" then generate "Makefile.in" from it.

Here's the "Makefile.am" we'll be using (We can't auto-generate it so you'll need to type it up yourself):

Listing of "Makefile.am":

```
bin_PROGRAMS=hello
hello_SOURCES=hello.c
```

bin_PROGRAMS tells what the name of our program(s) is, which is "hello" in our case.
And hello_SOURCES tells what source files are used to create our program "hello".
In our case, we only have one source file, "hello.c".
(You may want to create programs with multiple sourcefiles in the future. In such a case, separate the arguments with spaces.)

There are many more commands you can use in the "Makefile.am", such as commands to build libraries, go into subdirectories, etc.

Now, we'll Automake this file. This requires two steps:

automake - Executing this command will certainly produce many errors. We'll need to check each error and fix it.
aclocal - While dealing with step #1, we'll be adding some Automake macros into "configure.ac". This will freak out Autoconf because Autoconf won't understand these new macros. But we'll be able to calm Autoconf down by creating a file called "aclocal.m4" with the definitions of the newly added macros. "aclocal" will create this file for us automatically.

Dealing with errors

Now, let's run automake and see what errors we'll need to fix:

```
$ automake
configure.ac: 4: `automake requires `AM_CONFIG_HEADER', not
`AC_CONFIG_HEADER'
automake: configure.ac: `PACKAGE' not defined in `configure.ac'
automake: configure.ac: `VERSION' not defined in `configure.ac'
automake: configure.ac: required file `./install-sh' not found
automake: configure.ac: required file `./mkinstalldirs' not found
automake: configure.ac: required file `./missing' not found
automake: Makefile.am: required file `./INSTALL' not found
automake: Makefile.am: required file `./NEWS' not found
automake: Makefile.am: required file `./README' not found
automake: Makefile.am: required file `./COPYING' not found
automake: Makefile.am: required file `./AUTHORS' not found
automake: Makefile.am: required file `./ChangeLog' not found
configure.ac: 4: required file `./[config.h].in' not found
automake: configure.ac: AC_ARG_PROGRAM must be used in `configure.ac'
automake: configure.ac: AC_PROG_INSTALL must be used in `configure.ac'
$
```

The first error is easy. I'll simply change AC_CONFIG_HEADER to AM_CONFIG_HEADER in "configure.ac". Automake is just telling you here it prefers its version of the macro instead of Autoconf's version:

Listing of "configure.ac":

```
...
#AC_CONFIG_HEADER([config.h])
AM_CONFIG_HEADER([config.h])
```

As for the errors about PACKAGE and VERSION, this information can be added to "configure.ac" in several ways. The best way here is to add AM_INIT_AUTOMAKE(hello, 1.0) here, right after AC_INIT(...):

Listing of "configure.ac":

```
...
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AM_INIT_AUTOMAKE(hello, 1.0)
...
```

But in fact, this is one of the few standard macros you should know when you use Automake. The first argument to the AM_INIT_AUTOMAKE is the name of the package, and the second argument is the version of the package.

One can get "install-sh", "mkinstalldirs", and "missing" in "/usr/share/automake/". These are some standard scripts used by Autoconf and Automake.

As for "INSTALL", the document describing how to install the program, I can probably the copy the version in "/usr/share/automake/".

```
$ ls

AUTHORS    Makefile.am      autoscan.log  config.status*  hello.c

COPYING    Makefile.in      config.h      configure*      install-sh*

ChangeLog  NEWS             config.h.in   configure.ac    missing*

INSTALL    README           config.h.in~  configure.scan  mkinstalldirs*

Makefile   autom4te.cache/  config.log    hello*          stamp-h.in

$
```

So, here is how we can use Automake:

```
# Create "Makefile.am".
# Run "automake".
# Fix error messages.
# Run "aclocal".
# Autoconf everything.
# Configure, compile, and enjoy!
```

Create your sources. (Be sure to have a simple "Makefile" to help Autoconf autodetect things better.)
    # Run "autoscan" to generate "configure.scan".
    # Rename "configure.scan" to "configure.ac".
    # Run "autoheader" to generate "config.h.in".
    # Make your source portable by looking at "config.h.in". (We previously did this at a later step by reading "config.h", but we can do it in this step by referring to "config.h.in" instead.)
    # Create "Makefile.am".
    # Run "automake".
    # Fix errors and run "automake" again.
    # Run "aclocal".
    # Run "autoconf".
    # Configure, make, and run!