CS 35L

Software Construction Laboratory

Lecture 8.1

26th February, 2019

Logistics

- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Presentations for Assignment 10
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 7 is due on 3rd March, 2018 at 11:55pm

Final Exam Reminder

- Date: 17th March, 2019
- Day: Sunday
- Time: 3pm to 6pm
- Location: TBD
- Exam Format: Open book, open notes
  - No electronic devices: calculators, smartphones, smart watches, etc
- 50% of course grade

Review - Previous Lab

- Lifecycle of a C program
- Static Linking
- Dynamic Linking
  - Static and Dynamic Loading
  - Advantages & Disadvantages

Dynamic Linking

Two library types

- Static Libraries (.a):
  - Library of object code which is linked with, and becomes part of the application
- Dynamic Libraries (.so):
  - Static Loading: Dynamically linked at run time. The libraries must be available during compile/link phase. The shared objects are not included into the executable component but are tied to the execution.
  - Dynamic Loading: Dynamically loaded/unloaded and linked during execution using the dynamic linking loader system functions.

Library Naming Convention

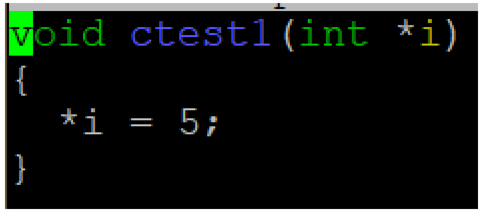
- Libraries are prefixed with lib. When linking the library name will not contain the library prefix
- Example: gcc src-file.c -lm –lpthread
- The libraries referenced during linking are
  - the math library ("m") found in /usr/lib/libm.a
  - the thread library ("pthread") found in /usr/lib/libpthread.a.

Static Libraries

- Use ar command to create a static library (ar stands for archive)
- Flags:
  - -c : create the archive
  - -v: verbose (shows filenames processed)
  - -q: quick append to the archive (without replacement)
  - -r: append to the archive with replacement
  - -t: display contents of archive
- For more info: Man ar
- Demo

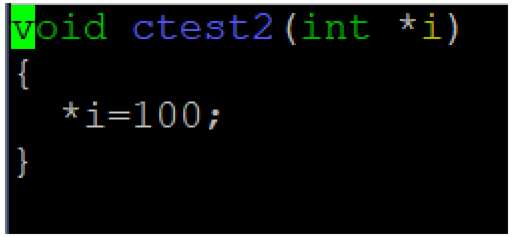
Static Library Demo

Ctest1.c



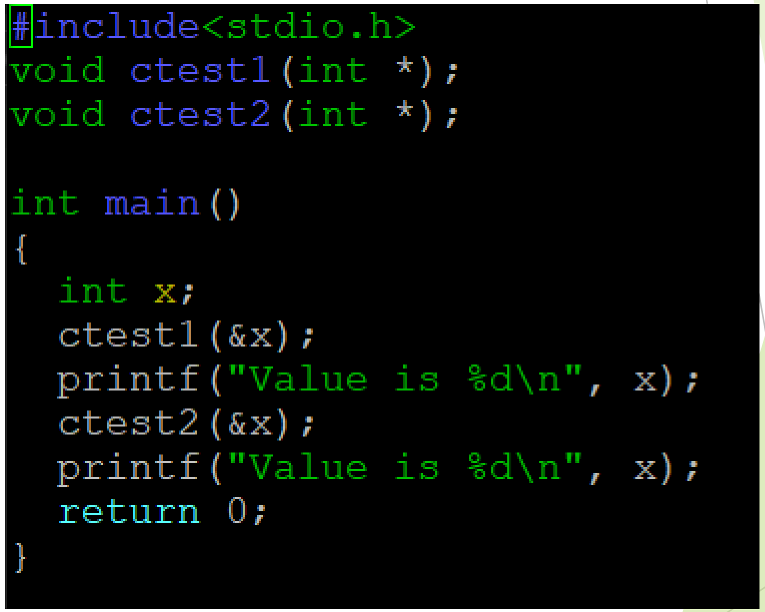
```
void ctest1(int *i)
{
  *i = 5;
}
```

Ctest2.c



```
void ctest2(int *i)
{
  *i=100;
}
```

Prog.c



```
#include<stdio.h>
void ctest1(int *);
void ctest2(int *);

int main()
{
  int x;
  ctest1(&x);
  printf("Value is %d\n", x);
  ctest2(&x);
  printf("Value is %d\n", x);
  return 0;
}
```

Commands:

1) gcc –c ctest1.c ctest2.c
2) ar -cvr libctest.a ctest1.o ctest2.o
3) ar -t libctest.a
4) gcc –o first.out prog.c libctest.a

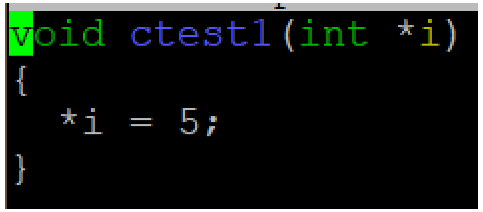
Dynamic Libraries

GCC Flags:

- -fPIC: Compiler directive to output position independent code, a characteristic required by shared libraries.
- -llibrary: Link with "liblibrary.a”
  - Without -L to directly specify the path, /usr/lib is used.
- -L: At compile time, find the library from this path.
- -Wl,rpath=.: -Wl passes options to linker.
  - -rpath at runtime finds .so from this path.
- -c: Generate object code from c code but do not link
- -shared: Produce a shared object which can then be linked with other objects to form an executable.
- https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html#Link-Options
- Reference: http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

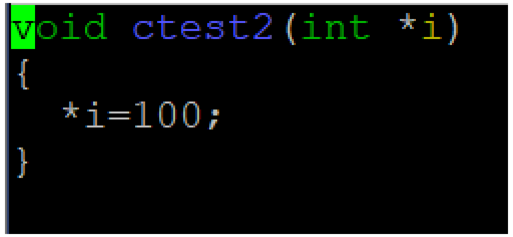
Dynamic Linking Demo

Ctest1.c



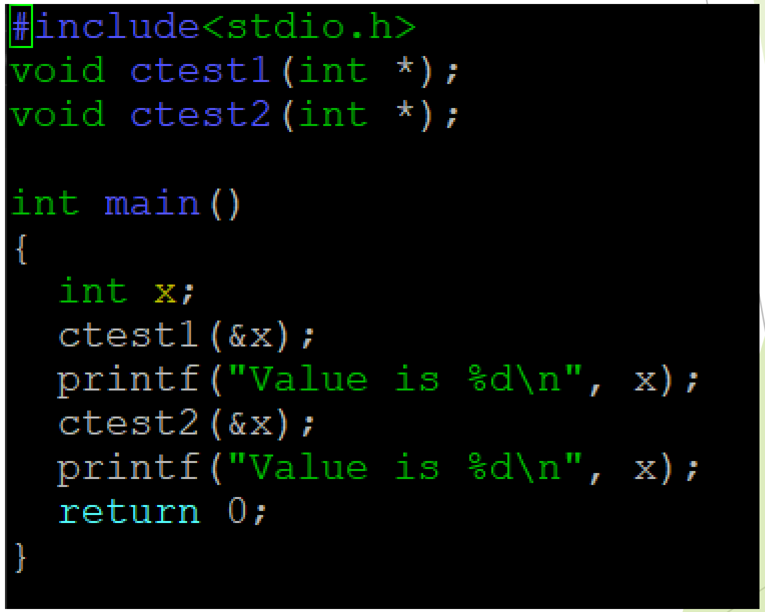
```c
void ctest1(int *i)
{
  *i = 5;
}
```

Ctest2.c



```c
void ctest2(int *i)
{
  *i=100;
}
```

Prog.c



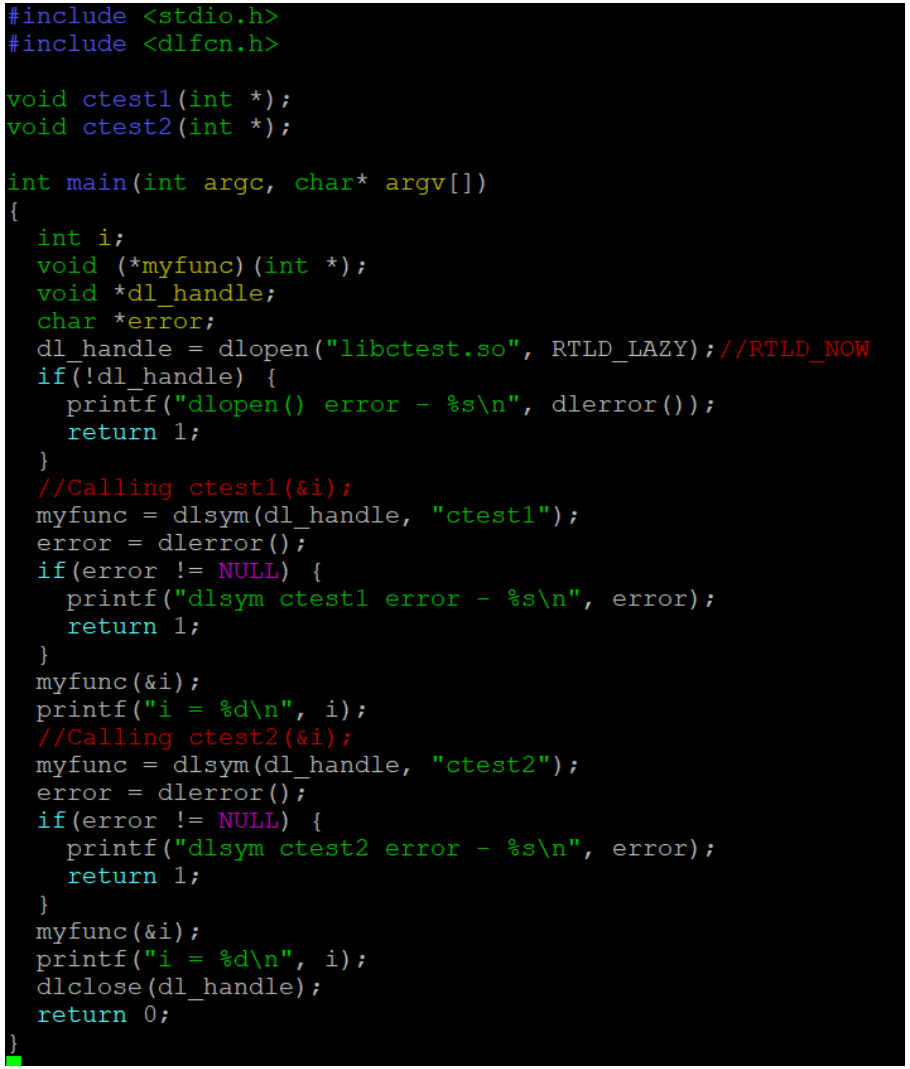
```c
#include<stdio.h>
void ctest1(int *);
void ctest2(int *);

int main()
{
  int x;
  ctest1(&x);
  printf("Value is %d\n", x);
  ctest2(&x);
  printf("Value is %d\n", x);
  return 0;
}
```

Commands:

1) gcc –Wall –fPIC –c ctest1.c ctest2.c
2) gcc –shared –o libctest.so ctest1.o ctest2.o
3) path=$PWD
4) gcc -Wall prog.c -lctest -o first.out (should give error)
5) gcc -Wall -L $path -lctest -o first.out
6) ldd first.out
7) ./first.out (If it does not execute, perform step 8)
8) export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$path
9) ./first.out

Dynamic loading



```
#include <stdio.h>
#include <dlfcn.h>

void ctest1(int *);
void ctest2(int *);

int main(int argc, char* argv[])
{
  int i;
  void (*myfunc)(int *);
  void *dl_handle;
  char *error;
  dl_handle = dlopen("libctest.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
    printf("dlopen() error - %s\n", dlerror());
    return 1;
  }
  //Calling ctest1(&i);
  myfunc = dlsym(dl_handle, "ctest1");
  error = dlerror();
  if(error != NULL) {
    printf("dlsym ctest1 error - %s\n", error);
    return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  //Calling ctest2(&i);
  myfunc = dlsym(dl_handle, "ctest2");
  error = dlerror();
  if(error != NULL) {
    printf("dlsym ctest2 error - %s\n", error);
    return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

- Emacs dynamic.c and input the code as shown in the left image
- path=$PWD
- Gcc –Wall –L $path dynamic.c –lctest –ldl –o dynamic.out
- You will have to set the environment variable LD_LIBRARY_PATH to include the path that contains libmymath.so
- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$path
- Man 3 dlopen – for flag descriptions

How are libraries dynamically loaded?

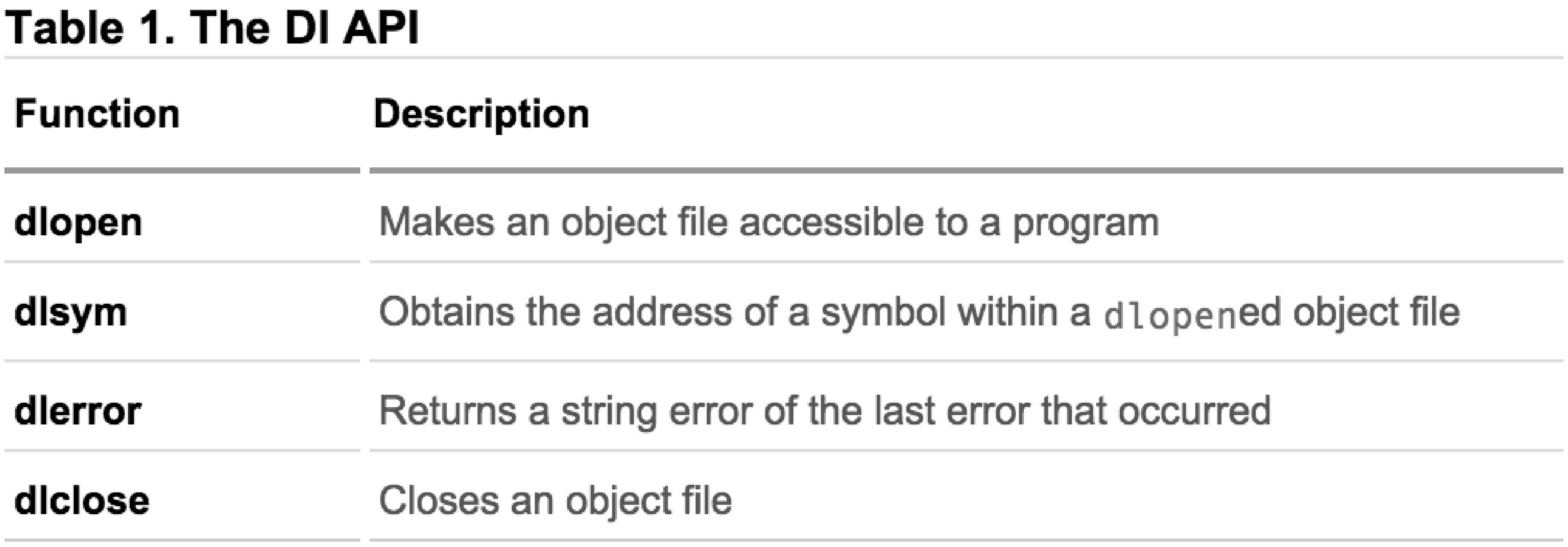


Table 1. The Dl API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dlopen`ed object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file

Attributes of Functions

- Used to declare certain things about functions called in your program
  - Help the compiler optimize calls and check code
- Also used to control memory placement, code generation options or call/return conventions within the function being annotated
- Introduced by the **attribute** keyword on a declaration, followed by an attribute specification inside double parentheses
- Reference: https://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Attributes.html

Attributes of Functions

- __attribute__ ((__constructor__))
  - Is run when dlopen() is called
- __attribute__ ((__destructor__))
  - Is run when dlclose() is called
- Example:

```
__attribute__ ((__constructor__))
void to_run_before (void) {
      printf("pre_func\n");
}
```

Assignment 7 - Laboratory

- Write and build simple cos(sqrt(3.0)) program in C
  - Use ldd to investigate which dynamic libraries your cos program loads
  - Use strace to investigate which system calls your cos program makes
- Use “ls /usr/bin | awk ‘NR%101==nnnnnnnnn%101’” to find ~25 linux commands to use ldd on
  - Record output for each one in your log and investigate any errors you might see
  - From all dynamic libraries you find, create a sorted list
    - Remember to omit the duplicates!

Assignment 7 – Home Work

- Split randall.c into 4 separate files
- Stitch the files together via static and dynamic linking to create the program
- randmain.c must use dynamic loading, dynamic linking to link up with randlibhw.c and randlibsw.c (using randlib.h)
- Write the randmain.mk makefile to do the linking

Assignment 7 – Home Work

- randall.c outputs N random bytes of data
- Look at the code and understand it
- main function
  - Checks number of arguments (name of program, N)
  - Uses helper function to check for HW support
  - Uses helper functions to generate random number using HW/SW
- Helper functions that check if hardware random number generator is available, and if it is, generates number
  - HW RNG exists if RDRAND instruction exists
  - Uses cpuid to check whether CPU supports RDRAND (30th bit of ECX register is set)
- Helper functions to generate random numbers using software implementation (/dev/urandom)

Assignment 7 – Home Work

- Divide randall.c into dynamically linked modules and a main program. Don’t want resulting executable to load code that it doesn't need (dynamic loading)
- randall.c = randcpuid.c + randlibhw.c + randlibsw.c + randmain.c
    - **randcpuid.c**: contains code that determines whether the current CPU has the RDRAND instruction. Should include randcpuid.h and include interface described by it.
    - **randlibhw.c:** contains the hardware implementation of the random number generator. Should include randlib.h and implement the interface described by it.
    - **randlibsw.c:** contains the software implementation of the random number generator. Should include randlib.h and implement the interface described by it.
    - **randmain.c:** contains the main program that glues together everything else. Should include randcpuid.h (as the corresponding module should be linked statically) but not randlib.h (as the corresponding module should be linked after main starts up). Depending on whether the hardware supports the RDRAND instruction, this main program should dynamically load the hardware-oriented or software-oriented implementation of randlib.

Assignment 7 – Home Work

- Create shared libraries
  - randlibsw.o : -fPIC, -c and other existing options
  - randlibhw.o : -fPIC, -c and other existing options
  - randlibsw.so : -shared option
  - randlibhw.so: -shared option
- Create library for static linking – 2 options
  - randcpuid.o: -c option, or
  - ar command to create an archive of static libraries
- Create object file for randmain
  - randmain.o: -c option
- Build randmain
  - randmain: -ldl -Wl,-rpath=${PWD}
  - If you used ar to create static library, use –lstaticlibrary option to statically link the library and optionally use –L option to specify the path for the statically linked library

Presentations

- Today's Presentation:
  - Renee Hsu
  - Atharv
- Next up:
  - Nathan Chen

Questions?