# CS29206: Systems Programming Lab

Autumn 2025

Creating and using Libraries

*Acknowledgement*

*Several slides, and contents of some others, have been taken from the slides made by Prof. Abhijit Das for an earlier offering of the course*

## Resources

- “An Introduction to gcc” by Brian Gough (Chapters 2-3)
- Manpage of ar command
- “Your Unix/Linux: The Ultimate Guide”, by Sumitabha Das (Chapter 16)

## Introduction

- A library is a pre-compiled archive of object files
  - These can be linked to user codes during compilation or during runtime
- Example: The math library consists of the following.
  - Definition of data types: float, double, . . .
  - Prototypes of Functions: pow, sqrt, atan, cosh, abs, . . .
  - Definition of Constants: M_PI, M_E, M_LOG2E, M_SQRT2, . . .
  - A precompiled archive of implementations of the math functions defined
- The first three are  in the header file math.h. You include it in your program #include <math.h>
- The fourth one, precompiled math library, is needed for linking to your final executable.
  - You specify the option -lm for this linking when you compile your C program

## Types of Libraries

- Static libraries
  - Prefix: *lib*
  - Extension: *.a*
  - Example: the static math library has the name *libm.a*
  - Functions from static libraries are actually inserted in the final executable during linking
    - Pro: the executable can run without needing anything else when it is run
    - Con: Since all the library code is added to your code, size of executable can be large
      - Example: Even if you call only one function fin your program from library with code for 100 functions, the entire code of all functions in the library will be added to the executable if you

## Types of Libraries

- Shared (or dynamic) libraries
  - Prefix: *lib*
  - Extension: *.so* (may be followed by another . And a version number)
    - Example: the dynamic math library has the name libm.so
  - Functions from dynamic libraries are not inserted in the final executable during linking, only pointers are inserted to where the code is
  - The actual functions are read from the library at runtime when the executable is run
    - Pro: the executable file sixe is small
    - Con: The executable file cannot be run as it is, the dynamic library must also be "loaded" at runtime

## Creating Static Libraries

## Building a static library

- Write your programs in .h and .c files as before
  - IMPORTANT: No main() function should be there in any of the C files
    - The main() function will be there in the C program that uses this library
- Generate .o files each of the C files
  - Compile using the -c option of gcc
- Combine all the .o files into a static library using the ar command
  - Must name the library as per the naming convention mentioned in the last slide

## Example

- Suppose you want to create  a static library named libstaque.a that will contain functions to implement stack and queue data structures
- Put all typedefs and function prototypes for stack and queue datatype are in defs.h, stack.h and queue.h respectively
  - See the files given
  - stack.h and queue.h includes defs.h, but we protect it with a #ifndef to make sure that any file (.h or .c) that includes both stack.h and queue.h includes def.h only once
- Put all functions bodies for stack and queue in stack.c and queue.c respectively
  - Do not put main() function in any file

## Example

- Create the .o files for stack.c and queue.c
- Use the ar (archive) command to create the library

```
$ gcc -c stack.c
$ gcc -c queue.c
$ ar rcs libstaque.a stack.o queue.o
```

```
$ ls -l
-rw-r--r-- 1 agupta faculty   138 Jul 31 16:35 defs.h
-rw-r--r-- 1 agupta faculty  5950 Jul 31 17:05 libstaque.a
-rw-r--r-- 1 agupta faculty   698 Jul 31 16:59 queue.c
-rw-r--r-- 1 agupta faculty   748 Jul 31 16:40 queue.h
-rw-r--r-- 1 agupta faculty  2680 Jul 31 17:04 queue.o
-rw-r--r-- 1 agupta faculty   894 Jul 31 17:03 stack.c
-rw-r--r-- 1 agupta faculty   617 Jul 31 16:37 stack.h
-rw-r--r-- 1 agupta faculty  2896 Jul 31 17:03 stack.o
-rw-r--r-- 1 agupta faculty   368 Jul 31 17:01 staquecheck.c
-rw-r--r-- 1 agupta faculty    38 Jul 31 16:40 staque.h
```

Options for ar command used:

r: insert with replacement
c: create the archive if not there
s: creates an index of the symbols defined

The order (options, then output file name, and the list of files to go in the library) is important.

# A peek inside the .o and .a files

A few lines not shown due to space

```
$ nm stack.o
0000000000000036 T destroystack
0000000000000076 T emptystack
                 U free
0000000000000000 T initstack
                 U malloc
00000000000000e7 T pop
                 U printf
0000000000000115 T printstack
00000000000000ab T push
                 U puts
0000000000000097 T top
```

```
$nm queue.o
0000000000000105 T dequeue
000000000000004a T destroyqueue
00000000000000ac T emptyqueue
00000000000000e4 T enqueue
                 U free
00000000000000c8 T front
0000000000000000 T initqueue
                 U malloc
0000000000000121 T printqueue
                 U puts
```

The "U" are undefined still!

Run nm libstaque.a and see what is printed

## Using the library created

- To use the library, a user program has to know the library name, and what header files to include for using the library
  - Example: To use the math library, you need to know the math library name and the header file to include (math.h)
- Create a header file libstaque.h that will include both stack.h and queue.h
  - This is the single header file that will be #included by any user program using the libstaque.a library
  - Similar to the math.h that you include when you want to use the math library
  - Not mandatory to have a single header file, just a convenience so that user does not have to remember the individual file names and which to include for which functions
  - Not always possible if there are large number of diverse types of functions in the library requiring many different header files
    - Example: C standard library. You include stdio.h for printf/scanf etc., stdlib.h for malloc, string.h for strcpy/strcmp etc.
    - Manpage of the function tells you which header file you need to include

## Example

- Suppose you write a C program in a file staquecheck.c that <u>declares</u> variables of type stack and queue and calls the stack and queue functions in the library
  - staquecheck.c will have a main() function
  - It will include the header file staque.h using #include
  - Compile it with -l option to create the final executable file a.out
    - gcc  staquecheck.c  -lstaque
    - Note that you didn't have to type the full name libstaque.a. The prefix lib and the suffix .a is automatically added by gcc
  - How will the compiler know in which directory libstaque.a is?
    - You can tell the compiler with the -L option of gcc (same format as for -I for .h files)
    - You can set the LIBRARY_PATH shell variable (same way as for C_INCLUDE_PATH for header files)

- So why no -L was needed when you used -lm in your programs?
  - There are default directories that the linker automatically looks at by default
    - Similar to what happens for #include <..>
  - math library libm.a is placed in one of these default directories, LIBRARY_PATH already set to them
- For using printf/scanf etc., which are in C library, you did not need to specify even -l option. Why?
  - C standard library is linked by default by gcc for C programs
- Actually, by default, dynamic libraries are used for both C standard library and math libraries. However, for them also, the above stands.

```
$ ls -l
-rwxr-xr-x 1 agupta faculty 17440 Jul 31 17:09 a.out
-rw-r--r-- 1 agupta faculty   138 Jul 31 16:35 defs.h
-rw-r--r-- 1 agupta faculty  5950 Jul 31 17:05 libstaque.a
-rw-r--r-- 1 agupta faculty   698 Jul 31 16:59 queue.c
-rw-r--r-- 1 agupta faculty   748 Jul 31 16:40 queue.h
-rw-r--r-- 1 agupta faculty  2680 Jul 31 17:04 queue.o
-rw-r--r-- 1 agupta faculty   894 Jul 31 17:03 stack.c
-rw-r--r-- 1 agupta faculty   617 Jul 31 16:37 stack.h
-rw-r--r-- 1 agupta faculty  2896 Jul 31 17:03 stack.o
-rw-r--r-- 1 agupta faculty   368 Jul 31 17:01 staquecheck.c
-rw-r--r-- 1 agupta faculty    38 Jul 31 16:40 staque.h
```

Note that every executable file that is linked with the library will have the full library code added to it.

## Creating Dynamic Libraries

## Creating dynamic library

- Consider that you want to build the same library, but this time a dynamic library libstaque.so
- Generate the .o files (stack.o and queue.o) for  the .c files (stack.c and queue.c) same as before, but with an extra option
  - gcc –fPIC –c stack.c
  - gcc –fPIC –c queue.c
  - -fPIC option forces generation of position-independent code (so the library can be loaded anywhere in memory, it is not dependent on being loaded at some specific memory area only)
- Use the command below to generate the library
  - gcc -shared -o libstaque.so stack.o queue.o

```
$ gcc -fPIC -c stack.c
$ gcc -fPIC -c queue.c
$ gcc -shared -o libstaque.so stack.o queue.o
```

```
$ ls -l
-rw-r--r-- 1 agupta faculty   138 Jul 31 18:04 defs.h
-rwxr-xr-x 1 agupta faculty 16816 Jul 31 18:10 libstaque.so
-rw-r--r-- 1 agupta faculty   698 Jul 31 18:04 queue.c
-rw-r--r-- 1 agupta faculty   748 Jul 31 18:04 queue.h
-rw-r--r-- 1 agupta faculty  2680 Jul 31 18:09 queue.o
-rw-r--r-- 1 agupta faculty   894 Jul 31 18:04 stack.c
-rw-r--r-- 1 agupta faculty   617 Jul 31 18:04 stack.h
-rw-r--r-- 1 agupta faculty  2896 Jul 31 18:09 stack.o
-rw-r--r-- 1 agupta faculty   368 Jul 31 18:04 staquecheck.c
-rw-r--r-- 1 agupta faculty    38 Jul 31 18:04 staque.h
```

## Using a shared library

- Use the same command as before
  - gcc staquecheck.c -lstaque
  - Use –L and –I as before as needed

```
$ ls -l
-rwxr-xr-x 1 agupta faculty 17016 Jul 31 18:17 a.out
-rw-r--r-- 1 agupta faculty   138 Jul 31 18:04 defs.h
-rwxr-xr-x 1 agupta faculty 16816 Jul 31 18:10 libstaque.so
-rw-r--r-- 1 agupta faculty   698 Jul 31 18:04 queue.c
-rw-r--r-- 1 agupta faculty   748 Jul 31 18:04 queue.h
-rw-r--r-- 1 agupta faculty  2680 Jul 31 18:09 queue.o
-rw-r--r-- 1 agupta faculty   894 Jul 31 18:04 stack.c
-rw-r--r-- 1 agupta faculty   617 Jul 31 18:04 stack.h
-rw-r--r-- 1 agupta faculty  2896 Jul 31 18:09 stack.o
-rw-r--r-- 1 agupta faculty   368 Jul 31 18:04 staquecheck.c
-rw-r--r-- 1 agupta faculty    38 Jul 31 18:04 staque.h
```

- Unlike the static library, the linker does not actually include the code of the library functions to the executable file created
- It only stores "pointer" to which library and where in that library a called function is
- The function code will be read from the .so library at runtime
- The stack and queue function show as undefined in your a.out created
- The .so library must be available from where you run your a.out at runtime for it to work
- If more than one program links with the same dynamic library, only one copy of the dynamic library will be loaded, all will access that one copy as needed.

```
$nm a.out | grep " U "
                 U destroyqueue
                 U destroystack
                 U enqueue
                 U exit@@GLIBC_2.2.5
                 U initqueue
                 U initstack
                 U __libc_start_main@@GLIBC_2.2.5
                 U printqueue
                 U printstack
                 U push
                 U rand@@GLIBC_2.2.5
```

## Trying to run the a.out

```
$ ./a.out
./a.out: error while loading shared libraries: libstaque.so: cannot open shared object file: No such file or directory
```

- What is the error? First run this command (ldd stands for "load dynamic")
  - $ ldd a.out

```
linux-vdso.so.1 (0x00007ffeda9d4000)
libstaque.so => not found
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe9e9609000)
/lib64/ld-linux-x86-64.so.2 (0x00007fe9e981d000)
```

  - It shows a.out needs 4 shared libraries, found 3, but could not find libstaque.so

- Why? Because you haven't yet told the system where to look for libstaque.so!
  - Same problems we solved at compilation time using –I and-L options, or C_INCLUDE_PATH and LIBRARY_PATH variables
  - But they address the problem at compilation time, not at runtime
- You need to set the environment variable LD_LIBRARY_PATH (LD is short form for "Load") to the directory to look for (in this example the current directory)

```
$ echo $LD_LIBRARY_PATH

$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
$ echo $LD_LIBRARY_PATH
:.
$ ./a.out
The contents of the stack (from top) is: 83  0
…
…
```

## What if the library changes?

- Ideally, all programs that have linked with the library should be recompiled and linked with the new library
- If you don’t
  - For static libraries, the executable already has the old library code in it, so it will continue to work without error, just will not see the new changes
  - For dynamic library,
    - The executable has the dynamic library identifier still in it, so if the old library is still available without any change, it will continue to wok without error, just will not see the new changes
    - If the old library is updated (name changed or contents of the library changed), you can get errors at runtime

# Forcing to use static libraries only

- You may have both static and dynamic versions of a library
  - Example: libm.a and libm.so for the math library
- What if you want to force the compiler to use only static libraries?
  - Use the –static flag in gcc
  - Will force the compiler to use only static libraries for all libraries being linked

```
$ gcc -static staquecheck.c -lstaque
$ ./a.out
The contents of the stack (from top) is: 83  0
…
…
```

```
$ ldd a.out
 not a dynamic executable
```

There is no dynamic libraries to load for a.out

```
$ nm a.out | grep " U "
$
```

Nothing is undefined in a.out

## Size difference between using static and dynamic libraries

Using dynamic libraries only

```
$ gcc staquecheck.c -lstaque
$ ls -l a.out
-rwxr-xr-x 1 agupta faculty 17016 Aug  4 12:20 a.out
```

Using static libraries only

```
$ gcc -static staquecheck.c -lstaque
$ ls -l a.out
-rwxr-xr-x 1 agupta faculty 877504 Aug  4 12:24 a.out
$ ldd a.out
        not a dynamic executable
```

- The huge increase in size is coming fro the fact that the -static option forced the compiler to use the static versions of the standard C library also, putting all code of it in a.out even though only printf is called from it

## An extreme example of size bloat

- Write a C file dummy.c with **only** these lines

```
int main()
{

}
```

- Now compile it linking with libstaque.h

```
$ gcc -static dummy.c -lstaque
$ ls -l a.out
-rwxr-xr-x 1 agupta faculty 871768 Aug  4 12:30 a.out
```

**See the a.out size, even though the .c file basically does nothing and did not use any function from any library!!!**

Compile without the -static flag to see the difference

## Practice in Lab

- Create a subdirectory under home directory named stq_library. Change to the subdirectory.
- Type the files defs.h stack.h queue.hstaque.h stack.c queue.c and staquecheck.c from the code given, all in the stq_library subdirectory
- Create the .o files for stack.c and queue.c only
- Combine the .o files into a static library named libstaque.a
- Compile staquecheck.c linking with libstaque.a to create an executable staquecheck.exe. Test to see it runs properly. Then delete it.
- Create a subdirectory called mylib under your home directory.
- Move (not copy) libstaque.a to ~/mylib
- Compile staquecheck.c again linking with libstaque.c to create an executable staquecheck.exe. Do you see any error?
- Use the -L option specifying the directory where libstaque.a is to compile without error
- Print the value of the shell variable LIBRARY_PATH
- Add the mylib directory created to it
- Now compile again without the –L option and test everything.

- Create another subdirectory named stq_shared under your home directory
- Copy all the .h and .c files from the stq_library subdirectory to it
- Create a dynamic library libstaque.so
- Compile staquecheck.c linking with libstack.so to create a.out
- Run ldd command on a.out to see that it depends on libstack.so
- Test out everything same as for the static library in the last slides
- Copy libstaque.so to the stq_library directory
- Compile staquecheck.c with and without the -static flag and observe the difference in size of a.out created