Preview

- What is LINUX
- What is GNU/LINUX Project
- Introduction to C Programming
- C System Environment
- C compilers in LINUX
- Linux System Roadmap
  - Header files
  - Libraries
    - Static Libraries
    - Shared Libraries

Story about Linux

- 1985 Professor **Andy Tanenbaum** (: University of Amsterdam) wrote a Unix like operating system from scratch, based on System V standards POSIX and IEEE, called **MINIX**.
- **Linus Torvald** ( ) wanted to upgrade MINIX and put in features and improvements, but Andrew Tanenbaum wanted **MINIX** the way it was and so Linus decided write his own kernel.
- He released **LINUX** (Linus Unix)on the Internet as an Open Source product and under his own license and then later in 1991.

Story about Linux

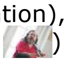
- **The Tanenbaum-Torvalds Debate by e-mail**

- **http://oreilly.com/catalog/opensources/book/appa.html**

Story about Linux (TUX)

- The idea of the penguin came from the creator of Linux, Linus Torvalds.
- The image was made by a man named Larry Ewing ( )in a competition to create a logo.
- The image, Tux, did not win, but it was picked as a mascot later.
- The first person called the penguin "Tux" was James Hughes who said that it stood for "(T)orvalds (U)ni(X)".

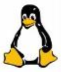
Story about GNU/LINUX

- The FSF (Free Software Foundation), started by **Richard Stallman**() on October 4, 1985
- The FSF started a project called GNU to fulfill this aim **GNU** stands for "**G**NU is **N**ot **U**nix"
- The **GNU** Project was launched in 1984 to develop a complete Unix-like operating system which is **free software**:

Story about GNU/Linux

- By 1991 GNU had already amassed a compiler (GCC), a C library, both very critical components of an operating system, and all associated generic Unix base programs but **not kernel**.
- The FSF naturally adopted the LINUX kernel to complete the GNU system - the GNU/LINUX operating system

Story about Linux



- There are hundreds numbers of operating systems platform are developed by using the Linux kernel. (i.e. SUSE, Red Hot, Ubuntu, Android …)
- The current full-featured version of kernel is 5.x.x and development continues.
- The latest stable version of the Linux kernel is: **stable version**: **5.4.15** 2020-01-26, check http://www.kernel.org

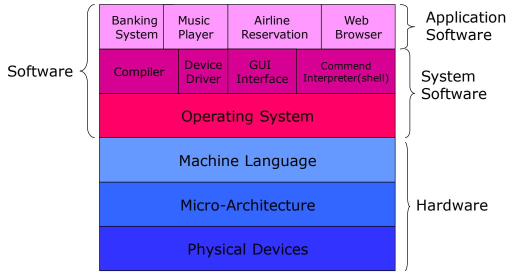
Linux Programs

- Linux applications are represented by two special types of files:
  - **Executable file** – directly executable by a computer.
  - **A Script** – collection of instruction for another program (interpreter)
- In Linux, we can replace scripts with compiled program (and vice versa).

Linux Architecture

- What is Operating System?
  - The software that **controls the hardware resources and provide an programming environment** under which programs can run (interface between HW and application SW).
  - We call this as **kernel**.
  - **System calls** is the layer of software interface to the kernel (unbuffered I/O) – it runs on kernel's space
  - **Library functions** are built on top of system call (Buffered I/O).
  - A **shell** is a command-line interpreter that read user input and execute command.

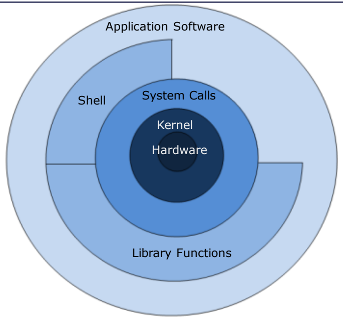
Linux Architecture



Linux Architecture

- Operating System's Basic 5 Layers
  - Process & thread management
  - Memory management
  - File management
  - Input / Output management
  - Deadlock management

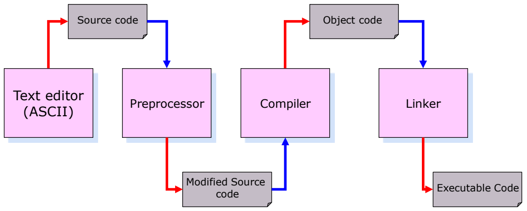
Linux Architecture



Introduction to C Programming

- **C** is a general-purpose, block structured, procedural, imperative computer programming language.
- It is developed in 1972 by **Dennis Ritchie** ( ) at the Bell Telephone Laboratories for use with the Unix operating system.
- C has greatly influenced many other popular programming languages, most notably C++, which originally began as an extension to C.

C System Environment



The C Compiler in LINUX

- LINUX provide several c compilers
  - c99 – POSIX systems
  - cc –UNIX Systems
  - gcc – GNU
- How to compile a program with gcc
  - Similar with g++ compiler
    1. gcc –c file.c //call compiler and create object code
    2. gcc file.o –o file //call linker and connect libraries to create executable code

    Or

    1. gcc –o file file.c //call compiler and linker to create executable code
  - Execute a program file
    - ./file which means execute the program in the current directory

Introduction to C

```
// Shows simple program
#include <iostream>
using namespace std;
int main ()
{
   cout << "Welcome to C++ ! \n";
   return 0;
}
```

```
/* A  simple program with C  */
#include <stdio.h>

int main ()
{
   printf("Welcome to C ! \n");
   return 0;
}
```

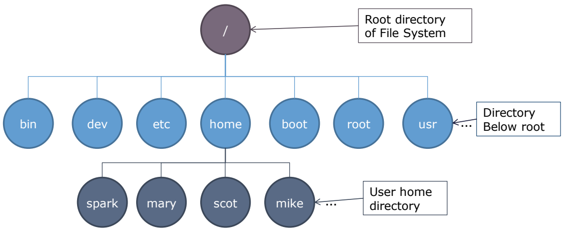
Introduction to C

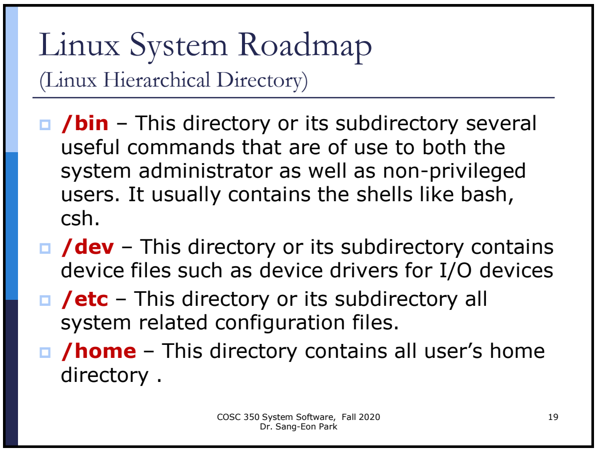
```
// twonum.cpp
#include <iostream>
using namespace std;

int main ()
{
     // declaration of variables
    int  a1,  a2,  sum;
    cout << "Enter first integer \n";
    cin >> a1;
    cout << "Enter second integer \n";
    cin  >> a2;
    sum = a1 + a2;
     cout << " Sum is  " << sum <<endl;
     return 0;
}
```

```
// twonum.c

#include <stdio.h>
int main ()
{
     /*declaration of variables */
    int a1, a2, sum;
    printf("Enter first integer \n");
    scanf("%d",&a1);
    printf ("Enter second integer \n");
    scanf("%d", &a2);
    sum = a1 + a2;
    printf (" Sum is %d \n", sum);
    return 0;
}
```

Linux System Roadmap

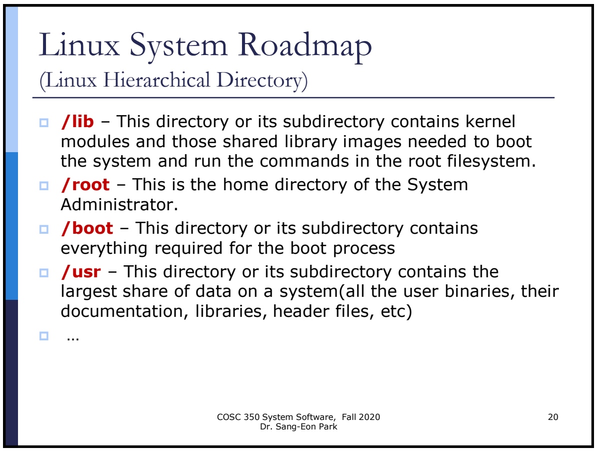
(Linux Hierarchical Directory)





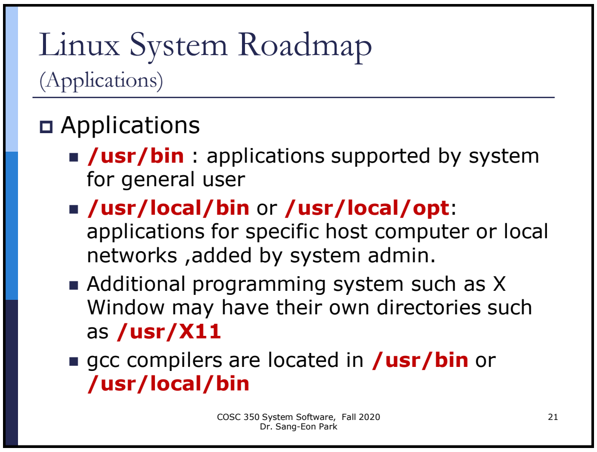
Linux System Roadmap
(Linux Hierarchical Directory)

- **/bin** – This directory or its subdirectory several useful commands that are of use to both the system administrator as well as non-privileged users. It usually contains the shells like bash, csh.
- **/dev** – This directory or its subdirectory contains device files such as device drivers for I/O devices
- **/etc** – This directory or its subdirectory all system related configuration files.
- **/home** – This directory contains all user's home directory .



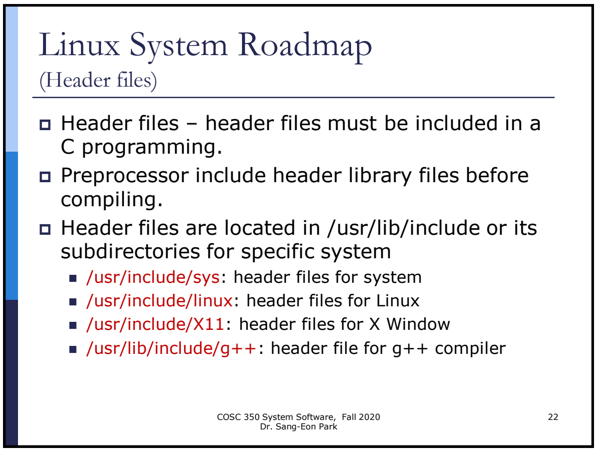
Linux System Roadmap
(Linux Hierarchical Directory)

- **/lib** – This directory or its subdirectory contains kernel modules and those shared library images needed to boot the system and run the commands in the root filesystem.
- **/root** – This is the home directory of the System Administrator.
- **/boot** – This directory or its subdirectory contains everything required for the boot process
- **/usr** – This directory or its subdirectory contains the largest share of data on a system(all the user binaries, their documentation, libraries, header files, etc)
- …



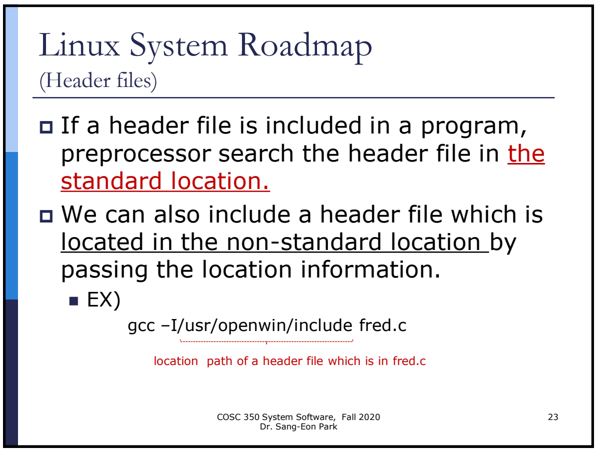
Linux System Roadmap
(Applications)

- Applications
  - **/usr/bin** : applications supported by system for general user
  - **/usr/local/bin** or **/usr/local/opt**: applications for specific host computer or local networks ,added by system admin.
  - Additional programming system such as X Window may have their own directories such as **/usr/X11**
  - gcc compilers are located in **/usr/bin** or **/usr/local/bin**



Linux System Roadmap
(Header files)

- Header files – header files must be included in a C programming.
- Preprocessor include header library files before compiling.
- Header files are located in /usr/lib/include or its subdirectories for specific system
  - /usr/include/sys: header files for system
  - /usr/include/linux: header files for Linux
  - /usr/include/X11: header files for X Window
  - /usr/lib/include/g++: header file for g++ compiler

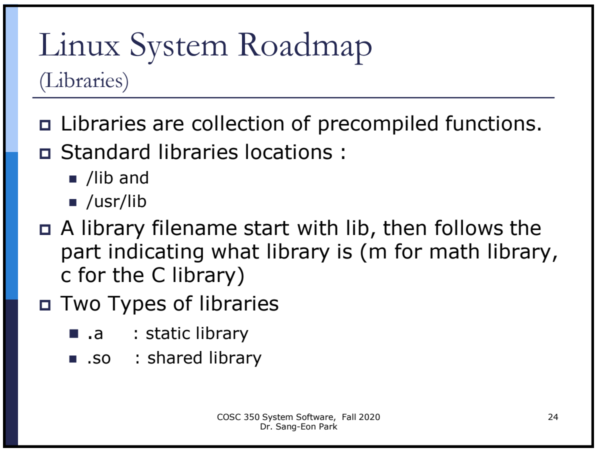


Linux System Roadmap
(Header files)

- If a header file is included in a program, preprocessor search the header file in the standard location.
- We can also include a header file which is located in the non-standard location by passing the location information.
  - EX)

```
gcc –I/usr/openwin/include fred.c
```

location path of a header file which is in fred.c



Linux System Roadmap
(Libraries)

- Libraries are collection of precompiled functions.
- Standard libraries locations :
  - /lib and
  - /usr/lib
- A library filename start with lib, then follows the part indicating what library is (m for math library, c for the C library)
- Two Types of libraries
  - .a : static library
  - .so : shared library

Linux System Roadmap
(Libraries)

- Linker knows the standard libraries locations.
- We can also instruct the linker to search a library at a specific location by passing the name of library with full path.
  - gcc –o some some.c /usr/lib/libm.a
- Shorted standard library location –lm means use libm.a static library
  - gcc –o some some.c -lm

Linux System Roadmap
(Libraries)

- We can also direct special directory by using –L flag where specific library is located.
  - gcc -o some -L/usr/openwin/lib some.c –laa

: means compile some.c with library libaa.so which is located in /usr/openwin/lib directory

Linux System Roadmap
(Static Libraries)

Static Library (call Archives)

- collection of object files in a ready to use form.
- To use a function in a Library, need include header file in your program.
- We can create and maintain our own static libraries by using **ar** (archive) program and compiling functions separately with gcc –c.

Linux System Roadmap
(How to Make Static Libraries)

- How to make your own static libraries?
  1. Make files for reusable functions.
  2. Create object codes by using commend
     - gcc –c file1.c (create file1.o)
     - gcc –c file2.c (create file2.o)
     - gcc –c file3.c (create file3.o)
  3. Create a header file (foo.h) which contains function prototypes for reusable functions. This header file must be included in the program where the reusable function is used.
  4. Create a library by using arc commend

Linux System Roadmap
(How to Make Static Libraries)

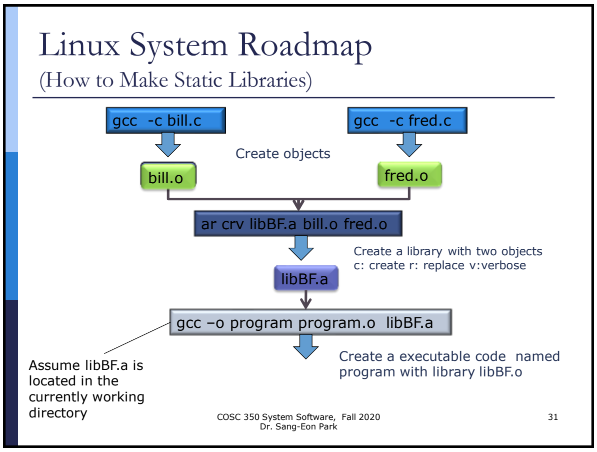
```
/* bill.c */
#include <stdio.h>

void bill (char *arg)
{
   printf("Bill: You passed %s\n ", arg);
}
```

```
/* A header file some.h
   for two function prototype
*/

void bill (char *);
void fred (int);
```

```
/*fred.c */
#include <stdio.h>

void fred (int arg)
{
    printf("Fred! You are passed: %d\n", arg);
}
```

Linux System Roadmap
(How to Make Static Libraries)

```
/* program.c   which include main function*/
#include " some.h"

int main ()
{
   bill(" Hello world ");
   fred (100);
   return 0;
}
```



Linux System Roadmap
(How to Make Static Libraries)

gcc -c bill.c

gcc -c fred.c

Create objects

bill.o

fred.o

ar crv libBF.a bill.o fred.o

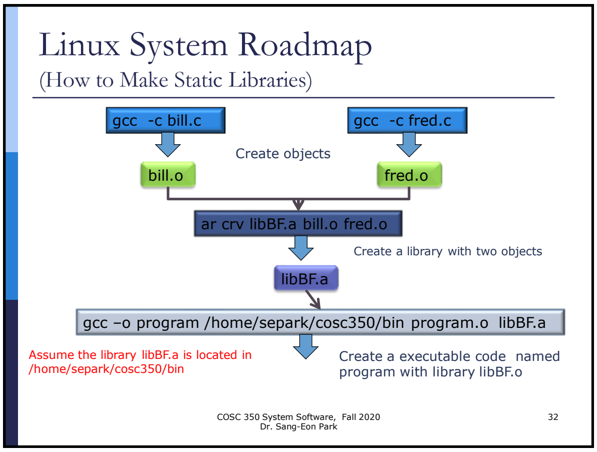
Create a library with two objects
c: create r: replace v:verbose

libBF.a

gcc –o program program.o libBF.a

Create a executable code named program with library libBF.o

Assume libBF.a is located in the currently working directory



Linux System Roadmap
(How to Make Static Libraries)

gcc -c bill.c

gcc -c fred.c

Create objects

bill.o

fred.o

ar crv libBF.a bill.o fred.o

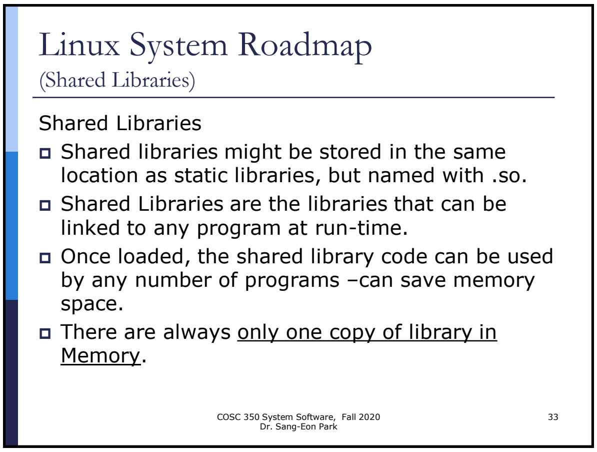
Create a library with two objects

libBF.a

gcc –o program /home/separk/cosc350/bin program.o libBF.a

Assume the library libBF.a is located in /home/separk/cosc350/bin

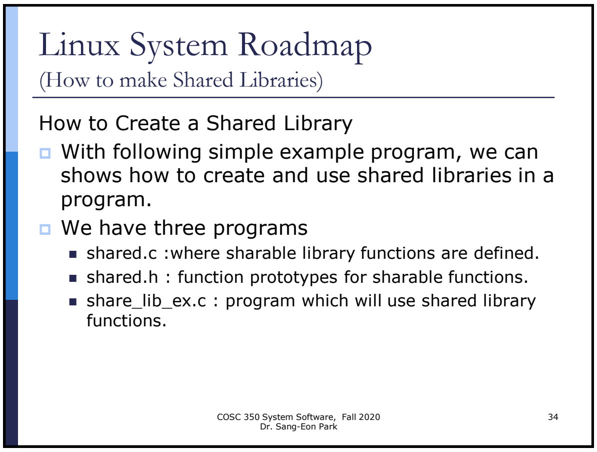
Create a executable code named program with library libBF.o



Linux System Roadmap
(Shared Libraries)

Shared Libraries

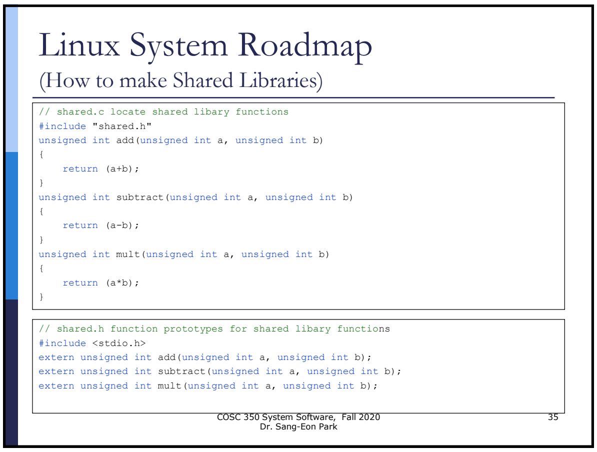
- Shared libraries might be stored in the same location as static libraries, but named with .so.
- Shared Libraries are the libraries that can be linked to any program at run-time.
- Once loaded, the shared library code can be used by any number of programs –can save memory space.
- There are always only one copy of library in Memory.



Linux System Roadmap
(How to make Shared Libraries)

How to Create a Shared Library

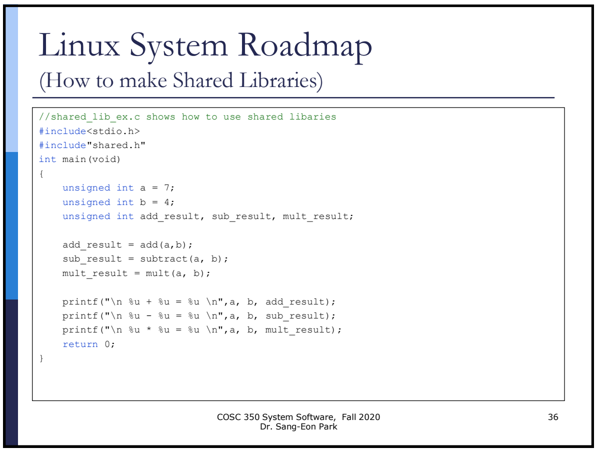
- With following simple example program, we can shows how to create and use shared libraries in a program.
- We have three programs
  - shared.c :where sharable library functions are defined.
  - shared.h : function prototypes for sharable functions.
  - share_lib_ex.c : program which will use shared library functions.



Linux System Roadmap
(How to make Shared Libraries)

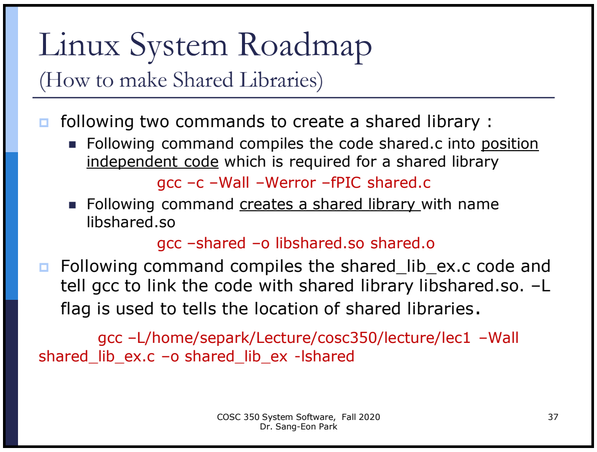
```
// shared.c locate shared libary functions
#include "shared.h"
unsigned int add(unsigned int a, unsigned int b)
{
    return (a+b);
}
unsigned int subtract(unsigned int a, unsigned int b)
{
    return (a-b);
}
unsigned int mult(unsigned int a, unsigned int b)
{
    return (a*b);
}
```

```
// shared.h function prototypes for shared libary functions
#include <stdio.h>
extern unsigned int add(unsigned int a, unsigned int b);
extern unsigned int subtract(unsigned int a, unsigned int b);
extern unsigned int mult(unsigned int a, unsigned int b);
```



Linux System Roadmap
(How to make Shared Libraries)

```
//shared_lib_ex.c shows how to use shared libaries
#include<stdio.h>
#include"shared.h"
int main(void)
{
    unsigned int a = 7;
    unsigned int b = 4;
    unsigned int add_result, sub_result, mult_result;

    add_result = add(a,b);
    sub_result = subtract(a, b);
    mult_result = mult(a, b);

    printf("\n %u + %u = %u \n",a, b, add_result);
    printf("\n %u - %u = %u \n",a, b, sub_result);
    printf("\n %u * %u = %u \n",a, b, mult_result);
    return 0;
}
```



Linux System Roadmap

(How to make Shared Libraries)

- following two commands to create a shared library :
  - Following command compiles the code shared.c into position independent code which is required for a shared library

```
gcc –c –Wall –Werror –fPIC shared.c
```

  - Following command creates a shared library with name libshared.so

```
gcc –shared –o libshared.so shared.o
```

- Following command compiles the shared_lib_ex.c code and tell gcc to link the code with shared library libshared.so. –L flag is used to tells the location of shared libraries.

```
gcc –L/home/separk/Lecture/cosc350/lecture/lec1  –Wall shared_lib_ex.c –o shared_lib_ex -lshared
```

COSC 350 System Software, Fall 2020
Dr. Sang-Eon Park

37