LINUX SYSTEM PROGRAMMING

The Core Operating System: The Kernel

- The term operating system is commonly used with two different meanings:
  - To denote the entire package consisting of the central software managing a computer's resources and all of the accompanying standard software tools, such as command-line interpreters, graphical user interfaces, file utilities, and editors.
  - More narrowly, to refer to the central software that manages and allocates computer resources (i.e., the CPU, RAM, and devices).

Tasks performed by the kernel

- Process Scheduling
- Memory Management
- Provision of a File System
- Access to devices.
- Networking
- Provision of a system call application programming interface (API).

Introduction To System Programming

- System software lives at a low level, interfacing directly with the kernel and core system libraries.
- The umbrella of system programming often includes kernel development.
- But our main focus will be on user-space system-level programming.

Three Corner Stones of System Programming on Linux

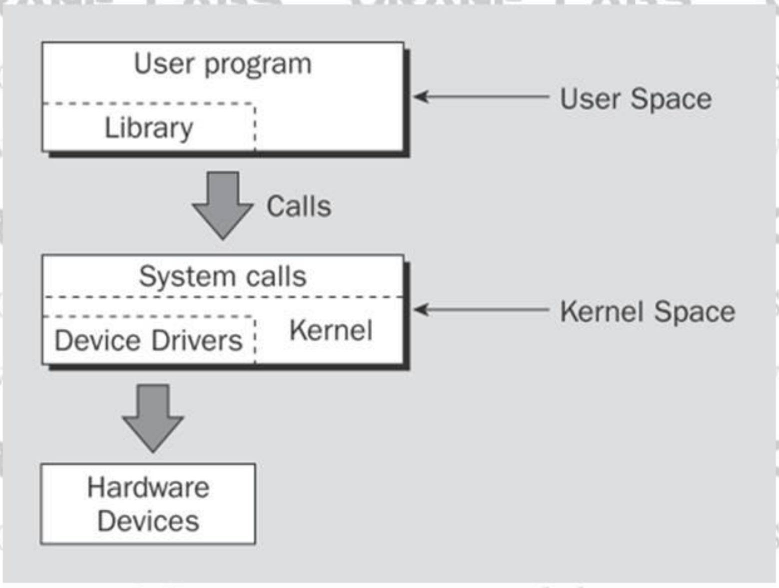
- System Calls
- The C Library
- The C Compiler

System Calls

- System programming starts with system calls.
- System calls are function invocations made from user space on—into the kernel in order to request some service or resource from the operating system.
- E.g. include read() and write().

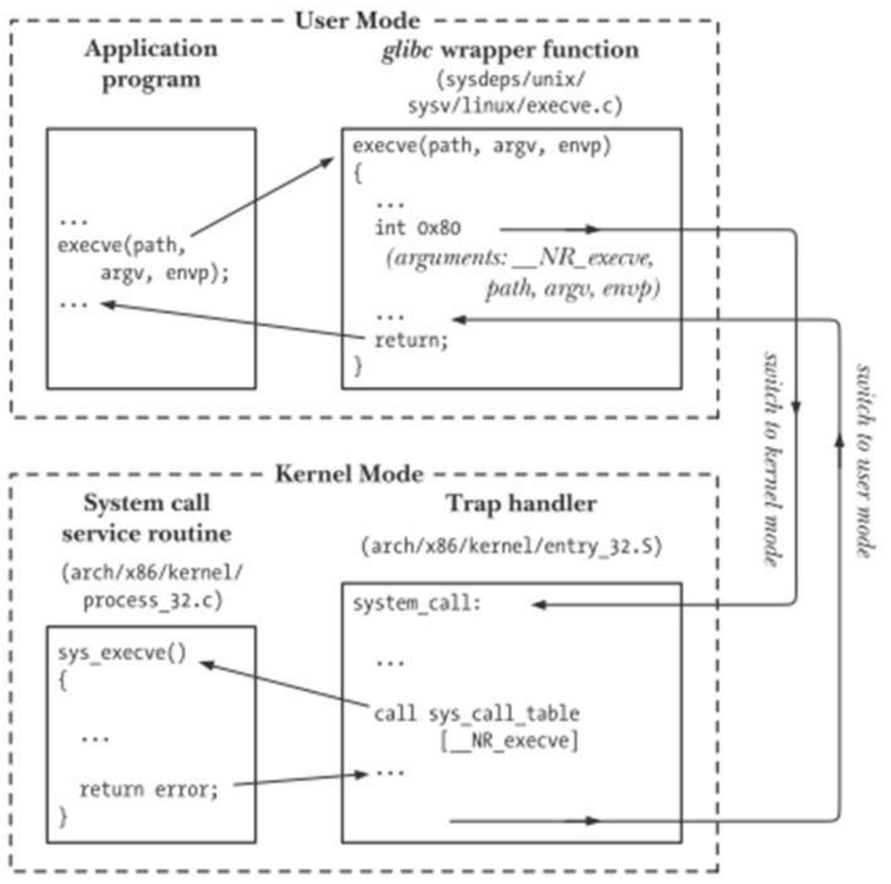
Some Points to Remember

- A system call changes the processor state from user mode to kernel mode, so that the CPU can access protected kernel memory.
- The set of system calls is fixed. Each system call is identified by a unique number.
- Each system call may have a set of arguments that specify information to be transferred from user space (i.e., the process's virtual address space) to kernel space and vice versa.



Process versus kernel views of the system

- A running system typically has numerous processes.
- A Process does not know the following:
  - When it will be next allocated the CPU
  - Does not know where it is located in the RAM
  - Similarly, a process doesn't know where on the disk drive the files it accesses are being held.
  - A process operates in isolation; it can't directly communicate with another process.
  - Finally, a process can't communicate directly with the input and output devices attached to the computer.


User Mode
Application program
...
execve(path, argv, envp);
...
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
execve(path, argv, envp)
{
...
int 0x80
(arguments: __NR_execve, path, argv, envp)
...
return;
}
switch to kernel mode
switch to user mode
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
sys_execve()
{
...
return error;
}
Trap handler
(arch/x86/kernel/entry_32.S)
system_call:
...
call sys_call_table [__NR_execve]
...

The C Library

- The C library (libc) is at the heart of Unix applications.
- Even when you're programming in another language, the C library is most likely in play, wrapped by the higher-level libraries, providing core services, and facilitating system call invocation.
- On modern Linux systems, the C library is provided by GNU libc, abbreviated as glibc.

The C Compiler

- In Linux, the standard C compiler is provided by the GNU Compiler Collection (gcc).
- Originally, gcc was GNU's version of cc, the C Compiler. Thus, gcc stood for GNU C Compiler.
- Over time, support was added for more and more languages. Consequently, nowadays gcc is used as the generic name for the family of GNU compilers.

Programs

- Programs normally exist in two forms.
  - Source code
  - Binary Executables

Writing your first C Program

```
#include <stdio.h>
int main()
{
printf("Hello World\n");
exit(0);
}
```

Compiling and Running our program

- For compiling we will use the following gcc command:
  - gcc –o First First.c
- For running the program:
  - ./First

DEVELOPMENT SYSTEM ROADMAP

Applications

- Applications are usually kept in directories reserved for them.
  - Applications supplied by the system for general use are found in the following directory:
    - /usr/bin
  - Applications added by system administrators for a particular system are in the following directory:
    - /usr/local/bin
  - The GNU compiler system's driver program, gcc, is typically located in
    - /usr/bin or /usr/local/bin

Header Files

- For programming in C and other languages, we need header files to provide definitions of constants and declarations for system and library function calls.
- For C these are almost located in:
  - /usr/include and its subdirectories.

Library Files

- Libraries are collections of precompiled functions that have been written to be reusable.
- Standard system libraries are usually stored in /lib and /usr/lib.
- A library filename always starts with lib.
- There are two types of libraries:
  - Static libraries with .a extension
  - and shared or dynamic libraries with .so extension.

Static Libraries

- Static libraries (sometimes also known as archives) were the only type of library on early UNIX systems.
- A static library is essentially a structured bundle of compiled object modules.
- To use them, you have to include them in you programs.
- The linker extracts copies of the required object modules from the library and copies these into the resulting executable file.
- We say that such a program is statically linked.

Disadvantages of Static libraries

- One is that the duplication of object code in different executable files wastes disk space.
- Second, if a library function requires modification, then, after recompiling that function and adding it to the static library, all applications that need to use the updated function must be relinked against the library

Creating Static libraries

- Lets create a library of two small functions one for addition and the other for multiplication:
- First source file add.c

```
#include <stdio.h>
void add(int arg1,int arg2)
{
Printf(The result is= %d\n", arg1 + arg2);
}
```

- The second source file mul.c

```
#include <stdio.h>
void mul(int arg1,int arg2)
{
Printf(The result is= %d\n", arg1 * arg2);
}
```

Compiling the two files

- Compilation is done by using the gcc compiler with the c option. which prevents the compiler from trying to create a complete program.
  - gcc –c add.c mul.c

Creating a Header file

- It is much better to declare your functions in a header file named lib.h

```
/*
This is lib.h. It declares the functions add and mul
   for users
*/
void add(int,int);
void mul(int,int);
```

Writing the main program

- program.c

```
#include “lib.h”
int main()
{
add(10,5);
exit(0);
}
```

Creating a Static library

-
- We use the ar program to create the archive and add our object files to it.
- The program is called ar because it creates archives, or collections, of individual files placed together in one large file.
  - $ ar crv libarth.a add.o mul.o

Using the Library

- Our library is now ready to use. We can add to the list of files to be used by the compiler to create our program like this:
  - $ gcc -o program program.o libarth.a
- We could also use the –loption to access our library, but as it is not in any of the standard places, we have to tell the compiler where to find it by using the –Loption like this:
  - $ gcc –o program program.o –L. –larth

Shared Libraries

- Shared libraries were designed to address the problems with static libraries.
  - If a program is linked against a shared library, then, instead of copying object modules from the library into the executable, the linker just writes a record into the executable to indicate that at run time the executable needs to use that shared library.
  - When the executable is loaded into memory at run time, a program called the dynamic linker ensures that the shared libraries required by the executable are found and loaded into memory.

Steps For creating a shared Library

- Using the same set of programs defined before.
-
- **Creating Object File with Position Independent Code:**
  - gcc -c –fPIC add.c –o add.o

Creating Shared library with the Object File

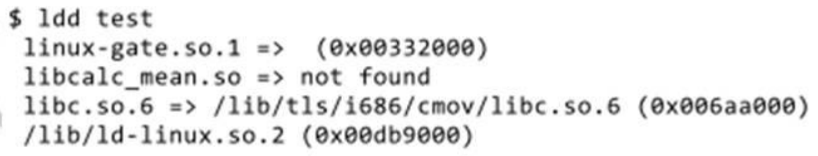
- Every shared library has a prefix "lib", the name of the library, the phrase ".so"
  - gcc -shared -o libarth.so add.o

Using the Shared Library

- Before using the library you will need to tell gcc where to find the library. This is done by specifying two options:
  - -l option tells the compiler to look for a file named libsomething.so The something is specified by the argument immediatelyfollowing the "-l". i.e. -lmean
  - -L option tells the compiler where to find the library.
- The Syntax
  - gcc -o program program.c -larth -L/home/ankr/direc

Checking the Executable

- Let us check if the path to our shared library is included successfully into the executable by linker as shown below:
  - ldd program



```
$ ldd test
  linux-gate.so.1 =>  (0x00332000)
  libcalc_mean.so => not found
  libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x006aa000)
  /lib/ld-linux.so.2 (0x00db9000)
```

Making the library available at run-time

- Using LD_LIBRARY_PATH
- We have to create or set the environment variable "LD_LIBRARY_PATH" to the directory containing the shared libraries.
  - export LD_LIBRARY_PATH=/home/cf/lib
- If in current directory you can give the following command
  - export LD_LIBRARY_PATH=.