Introduction to Unix Operating System

Operating System?

- Provides an interface between user and the hardware.
- Interface in terms of ??
- File management, I/O and network management, execution of the programs, etc.
- Examples: Windows, Linux, Unix, OS X, Vxworks, ThreadX, ucos, etc.
- Resource manager
- controls the hardware resources of the computer and provides an environment under which programs can run.

Types of OSs

- Classified based on types of tasks it performs
  - Single user OS
  - Multi user OS
  - Single tasking
  - Multitasking
  - Real time OS

Hardware Resources

- CPU
- Memory –main & secondary
- I/O Devices

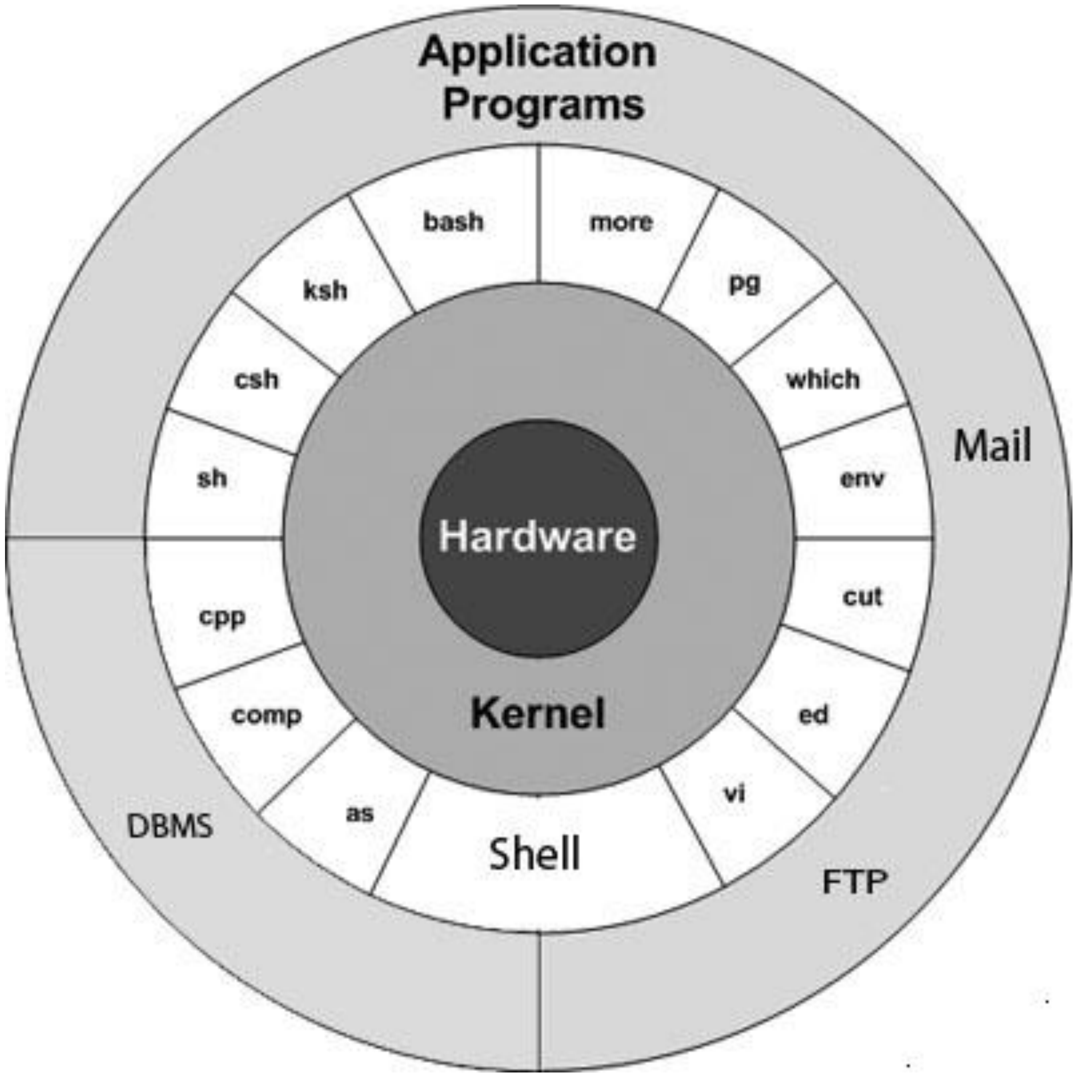
Unix OS

- Multi user multitasking OS
- Developed in C programming language at AT&T Bell Laboratories during 1970s
- Widely used and adapted
- Considered best operating system for enhancing the theoretical concepts of Operating System

The UNIX Operating System

- Written in high-level language
- Simple user interface that can provide the services user wants
- Allows complex programs to be built from simpler programs
- Hierarchical file system allowing easy maintenance
- Multi-user multi-process system; each user can execute several processes
- Hides machine architecture from the user

Layered architecture(UNIX OS)



Kernel

- Kernel is a program that constitutes the central core of the Operating System.
- Controls the h/w and provides an environment to execute a program
- Provides basic services to all other parts of the OS including
  - Management of Process, Memory, File, I/O, etc.
- Cannot directly interact with the user but, interacts with Shell program to control the H/W (Processor, memory, disk drives etc.).

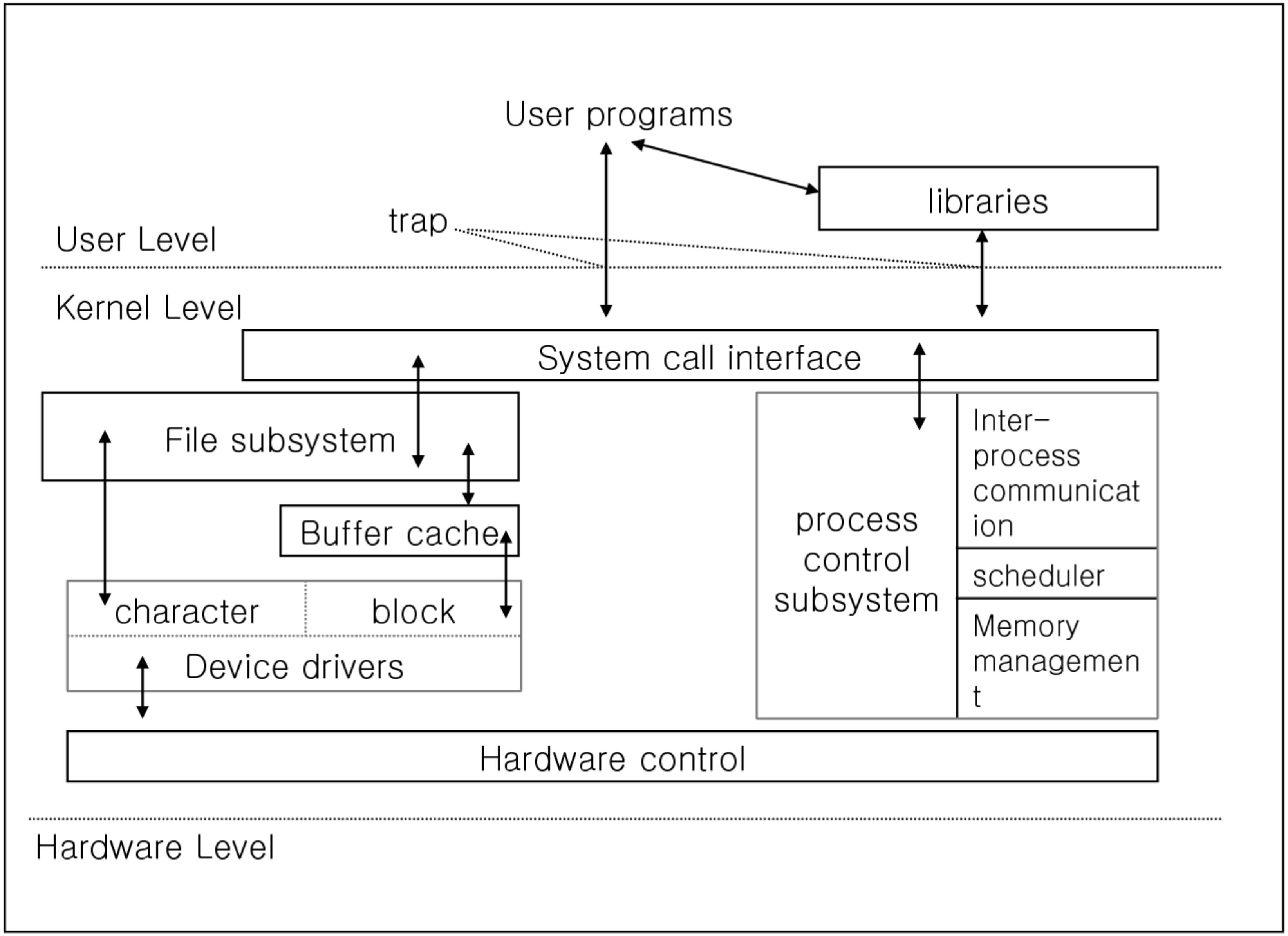
Kernel..

- These services are requested by other parts of OS or by the application program through a set of program interfaces called as System Call.
- In brief, the set of system calls and the internal algorithms that implement them form the body of the kernel

Shell

- Shell is a program that provides text-only interface to the user.
- It primary function is to read commands from the console and execute them.
- The term Shell comes from the fact that it is the outer most part of the OS.
- It is an interface between user and kernel.

Block Diagram of the System Kernel



- Major components of kernel
  - ✓ File System sub-module
    - It manages files, allocating file space, administrating free space, controlling access to files and retrieving data for user.
  - ✓ Process control sub-module
    - It is responsible for process synchronization, inter-process communication, memory management and process scheduling.
- The file sub-system and the process control subsystem interact when loading a file into memory before execution. The process subsystem reads executable files from the memory before executing them.

- Memory management module controls the allocation of memory. If at any time, system does not have enough main memory for all processes, the kernel moves them from main memory to secondary memory so that all processes get a fair chance to execute.

- Hardware control is responsible for handling interrupts and for communication with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. Kernel services the interrupt and resumes the execution of interrupted process.

System Calls

What is a system call?

- OS is a bridge between the user and hardware
- User program will not directly interact with the underlying hardware
- Instead the OS will execute these requests on behalf of the user program
- System call: Is a request for the OS to do something on behalf of the user's program.

System Call…

- For example,
  - File Management system calls – open(),read(),write(), close().
  - Process Management –

    fork(), exec()

System Call…

- The system calls are functions used in the kernel itself
- A system call executes code in the *kernel*, there must be a mechanism to change the mode of a process from *user mode* to *kernel mode* (*interrupt*)
- UNIX *system calls* are used to manage the file system, control processes, and to provide inter process communication.

GENERAL CLASS	SPECIFIC CLASS	SYSTEM CALL
File Structure Related Calls	Creating a Channel	creat()
		open()
		close()
	Input/Output	read()
		write()
	Random Access	lseek()
	Channel Duplication	dup()
	Aliasing and Removing Files	link()
		unlink()
	File Status	stat()
		fstat()
	Access Control	access()
		chmod()
		chown()
		umask()
	Device Control	ioctl()

```c
#include <stdio.h>
#include <unistd.h> /* contains fork prototype */
int main(void)
{
printf("Hello World!\n");
fork( );
printf("I am after forking\n");
printf("\t I am process %d.\n", getpid( ));
}
```

OUTPUT:

```
Hello World!
    I am after forking
    I am process 5502.

    I am after forking
    I am process 5503.
```

Interrupt

- An **interrupt** is a signal to the processor emitted by H/W or S/W indicating an event that needs immediate attention.
- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current process the processor is executing.
- The processor responds by suspending its current activities, saving its state, and executing a small program called an *interrupt handler* (or interrupt service routine, ISR) to deal with the event.
- This interruption is temporary, and after the interrupt handler finishes, the processor resumes execution of the previous process.

Interrupt

- **Program interrupt** : It is generated by illegal instructions in the program.
  - Division by zero, arithmetic overflow, reference outside user’s space, etc.
- **Timer interrupt:** Generated by clock within the processor.
  - When the time slot of the processor expired, the OS sends the interrupt signal to processor to suspend the current job.

Interrupt

- **I/O interrupt** : It is generated by I/O controller.
  - If CPU issues a Read command to I/O module, the I/O module is ready at that time, then I/O module sends the interrupt signal to CPU about the status.

- **H/W interrupt:** this type of interrupt generated by H/W.
  - Any H/w component may be fail
  - Memory failure, power failure.

Need for System Call

- **Protection:** A user process when given direct access to perform privileged operations, may unintentionally\maliciously corrupt other user’s data or the system data. The OS does the policing job to make sure that every user’s data and resources are safe from corruption by other users.

- **Abstraction:** A user program does not have the burden of knowing how to interact with the hardware. It can just say that a content of some file, “xyz.txt” is needed. The actual placing of the file in physical memory is done by the OS.

How do system calls work?

- Machines have two kinds of instructions
  1. "*normal*" instructions, e.g., add, sub, etc.
  2. "*privileged*" instructions, e.g.,

     initiate I/O

     switch state vectors or contexts

     load/save from protected memory, etc.
- OS hide *privileged instructions* and provide *virtual instructions* to access and manipulate virtual resources, e.g., I/O to and from disc files
- Virtual instructions are *system calls*
- Operating systems *interpret* virtual instructions

Process modes

- Machine level typically has 2 modes, e.g., “user mode” and “kernel mode”
- *User mode*
  - processor executes “normal” instructions in the user’s program
  - upon encountering a “privileged” instruction, processor switches to kernel mode, and the OS performs a service
- *Kernel mode*

  processor executes both normal and privileged instructions
- User-to-kernel switch saves the information necessary to continue the execution of the user process
- Another view

  OS is a process that runs in kernel mode.

Example

- Read the name of a file to be edited from the user (USER MODE)
- Pass this file name to kernel using a system call (KERNEL MODE)
- Kernel checks to see if the file exists in the system and informs the user program using return value of the system call(KERNEL MODE)
- Pass the data to be written to the file using a system call (USER MODE)
- Write data in the disk portion where that file resides and inform user if successfully written using return value of the system call (KERNEL MODE)

When system call occurs…

- control switches from user mode to kernel mode
- some operation is performed in kernel mode
- Then control again switches back from kernel mode to user mode
- switch between modes is also called context-switches and every system call needs 2 context switches

Deeper into system calls

C Library Function Call & System Call

- Functions defined in different C library are used in C program. Eg: malloc(), printf(),etc.
- These functions internally invoke system calls to get the service from the kernel.
- C library function call **CANNOT** invoke the functionalities in kernel.
- User application can directly invoke system call or through C library function.
- For example, printf() in turn calls a write() system call to print a string to stdout.

C Library Function Call & System Call

- malloc() in C is a library function in <stdlib> which is used to allocate memory. This in turn invokes a system call sbrk() which increases or decreases the address space of the process by the specified number of bytes.

- System calls like fork(), exec() can be directly invoked by the program.

Environment Variable

- Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.
- Environment variable create a operating environment for a running process.
- Example environment variables: PATH,HOME etc.

Program Execution Process

- An executable program is executed with the help of a system call exec().
- This system call loads the text and data of a program into the memory before execution.
- This system call takes the name of the executable along with command line arguments as input parameter and invokes the main() function of the program.
- Along with command line arguments, it also takes the list of environment variables.

Memory Layout of a C program

- Program is loaded along with following details in to the memory
  - Text Segment: Machine instructions to be executed by CPU
  - Data Segment: Initialized Data, Un-initialized Data
  - Stack Space: for statically allocated data during execution
  - Heap Space: for dynamically allocated memory
  - Command line arguments and environment variables

User Stack vs Kernel Stack

- Like user stack, kernel stack is also maintained by each process.
- Kernel Stack stores the required information during the execution of a system call invoked by some library function call.

Operating System Services

- File Management: Required to open, read, write or close operations of file or directory.
- Process Management: Required for executing one or many programs.
- CPU Scheduling: Required to schedule different processes to execute on CPU.
- I/O Management: Required to manage input-output operations.

File Management

- It handles File related operations like open, close, read and write.
- These operations are handled by different system calls.
- These system calls can be directly invoked by the application code or indirectly invoked by the library functions.

Process Management

- It handles creation and execution of Processes
- Handles Process Synchronization
- Inter-Process Communication
- There are different system calls to handle services related to process management.
- For example: fork(), exec(), wait(), mutex() etc.

CPU Scheduling

- There are various algorithms to implement scheduling of processes.
- These are implemented as part of the OS software.
- For example, Linux Scheduler: Completely Fair Scheduler.