NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

Assignment – 01

Name	: Joy Kumar Ghosh
Student ID	: 2211424 6 42
Course No.	: CSE 323
Course Title	: Operating Systems Design
Section	: 7
Date	: 16 June 2024

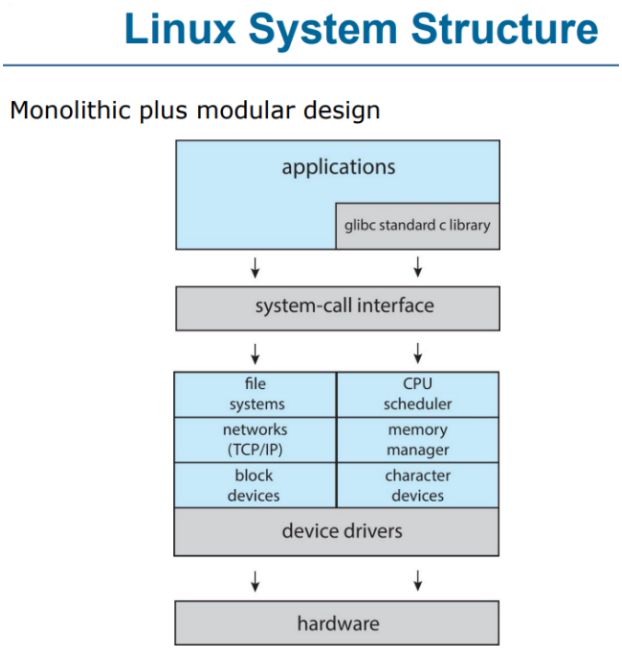
Linux Operating System

An operating system is a program that is an intermediary between a computer user and the computer hardware. The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system.
Operating System Structure can be designed in many ways:

1. Simple structure – MS-DOS
2. More complex - UNIX
3. Layered – an abstraction
4. Microkernel – Mach
5. Modular
6. Hybrid

Linux Kernel Management:

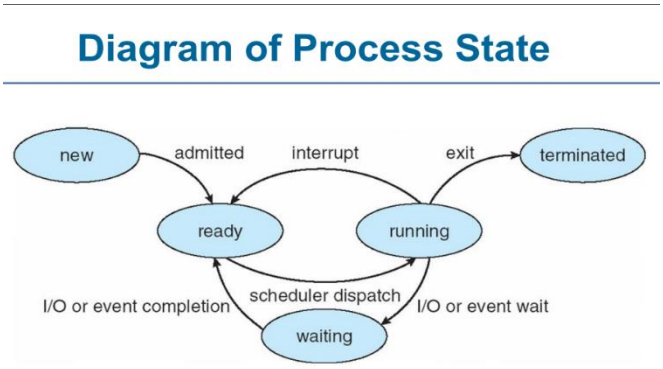
The Linux operating system is based on UNIX. Applications typically use the Glibc standard C library when communicating with the system call interface to the kernel. The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space, but it does have a modular design that allows the kernel to be modified during run time.



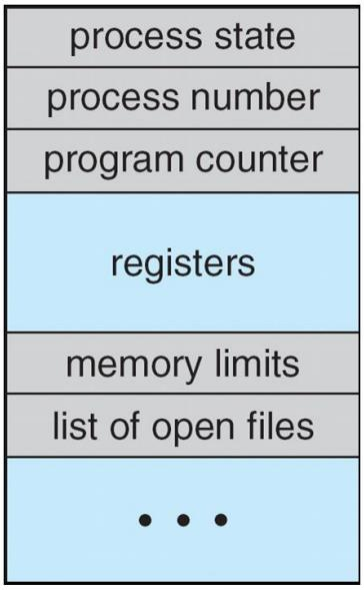
Process Management:

The operating system is responsible for the following activities in connection with process management such as: creating, deleting, suspending, and resuming process, also, mechanisms for process synchronization, communication, and deadlock handling.

As a process executes, it changes state within new, running, waiting, ready, and terminated.



Process Control Block (PCB) stores all the information associated with each process, like, process state, program counter, CPU registers, CPU scheduling information, Memory management information, accounting information about CPU used, clock time elapsed, time limits, and I/O status information.

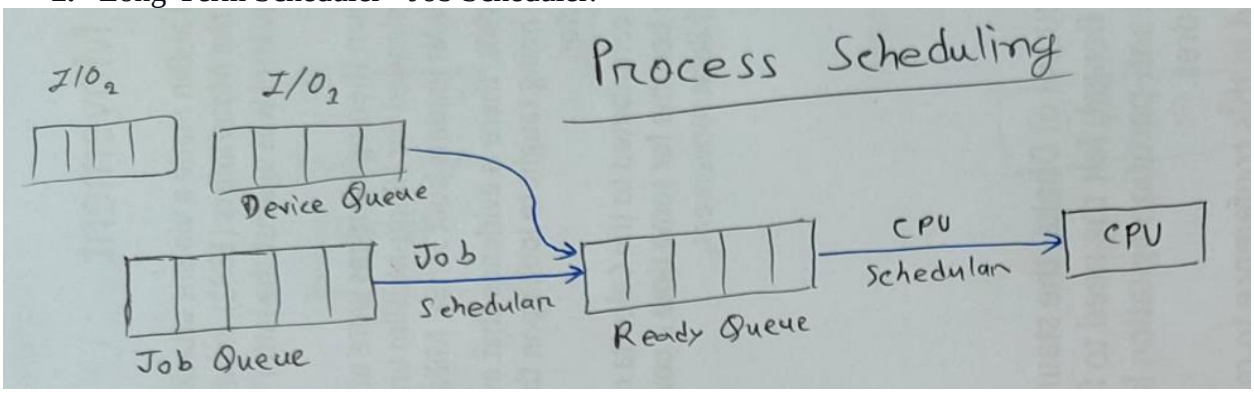


Process Scheduling:
To maximize the CPU usages, we need to switch processes onto the CPU for time-sharing quickly, and the process scheduler does this by selecting among available processes for the subsequent execution on the CPU.
There are three types of Queues:

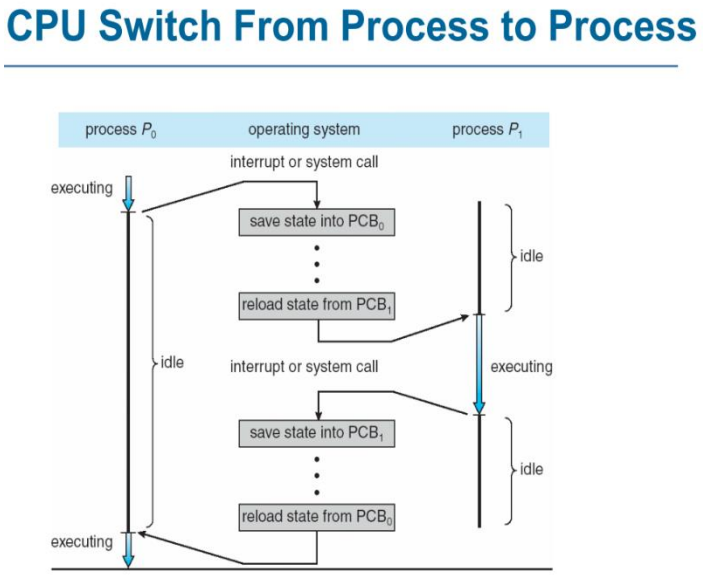
1. Job Queue: contains all processes in the system.
2. Ready Queue: includes processes that are ready and waiting to be executed.
3. Device Queue: all processes that need I/O operations.

There are two types of schedular:

1. Short-Term Schedular - CPU Schedular, executed too frequently.
2. Long-Term Scheduler - Job Scheduler.

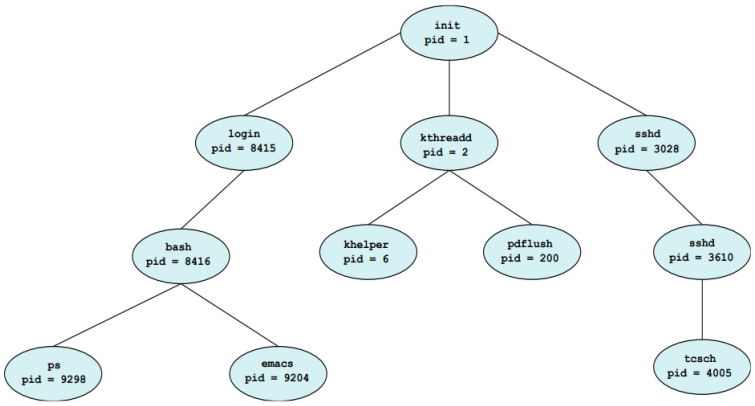


Context Switch:
When the CPU needs to switch to another process, the system must save the state of the old process and load the saved state for the new process via a context switch. Therefore, the context switch will execute between process switches.



Process Creation:
Parent processes create children's processes, which, in turn, create other processes, forming a tree of processes. These processes are identified and managed via a process identifier (PID). They may or may not share resources among themselves.

A Tree of Processes in Linux



Memory Management:

To execute a program, all (or part) of the instructions must be in memory. Also, all (or part) of the data the program needs must be in memory.
Memory management activities

1. Keeping track of which parts of memory are currently being used and by whom.
2. Deciding which processes (or parts thereof) and data to move into and out of memory.
3. Allocating and deallocating memory space as needed.

Memory Allocation:
The main memory must accommodate the operating system and the various user processes. We, therefore, need to allocate the main memory in the most efficient way possible.

Swapping:
A process must be executed in memory. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. Swapping makes it possible for the total physical address space of all processes to exceed the actual physical memory of the system, thus increasing the degree of multiprogramming in a system.

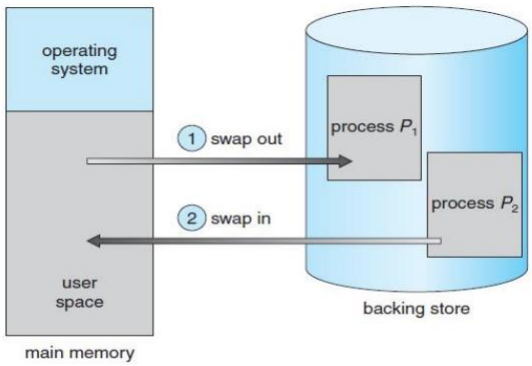
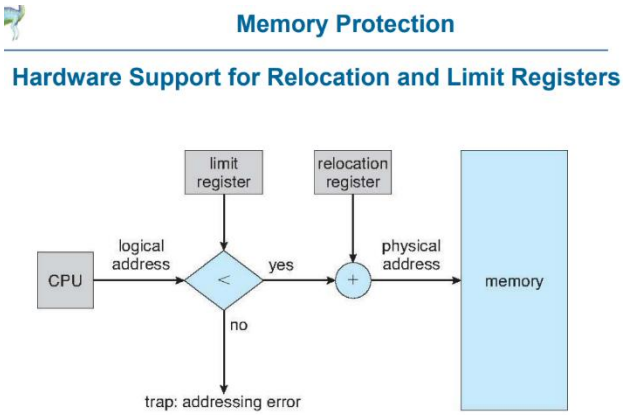


Figure 8.5 Swapping of two processes using a disk as a backing store.

Contiguous Memory Allocation:
In contiguous memory allocation, the main memory is divided into two partitions: one for the operating system, typically placed in low memory due to the location of the interrupt vector, and one for user processes. This method efficiently allocates memory to accommodate the OS and multiple user processes simultaneously. Each user process resides in a single contiguous section of memory. The system must decide how to allocate these sections to the processes waiting in the

input queue, ensuring that each process is placed in a contiguous block of memory adjacent to the following process.

Memory protection:



Memory protection prevents processes from accessing memory they do not own by using a relocation register and a limit register. The relocation register holds the smallest physical address, while the limit register specifies the range of logical addresses. When the CPU generates an address, it is checked against these registers to ensure it falls within the allowed range, protecting the operating system and other processes. The Memory Management Unit (MMU) maps logical addresses by adding the relocation register's value. This scheme allows the OS size to change dynamically, accommodating transient code, such as device drivers, which can be loaded and unloaded as needed.

Memory allocation:

Memory allocation involves dividing memory into partitions for processes. Initially, fixed-sized partitions limited multiprogramming, with each holding one process. A more flexible method, the variable-partition scheme, treats memory as holes of various sizes. Processes are allocated blocks based on requirements, and memory is merged when processes terminate.

Allocation strategies include:

- **First fit**: Use the first sufficiently large hole.
- **Best fit**: Use the smallest adequate hole.
- **Worst fit**: Use the largest hole.

Simulations show that the first and best fit are more efficient than the worst, with the first fit generally being faster.

Fragmentation:

Both first-fit and best-fit memory allocation strategies suffer from external fragmentation, where free memory is fragmented into small, noncontiguous pieces. This makes it challenging to allocate memory efficiently, potentially wasting significant space. Internal fragmentation occurs when allocated memory exceeds the requested amount, leaving small, unusable gaps within partitions. Solutions include compaction, which consolidates free memory into one block but is only possible with dynamic relocation and can be costly. Noncontiguous memory allocation methods like segmentation and paging can also address fragmentation by allowing physical memory to be allocated wherever available.

Segmentation:
Segmentation is a memory-management scheme that aligns with a programmer's view of memory as a collection of variable-sized segments rather than a linear array. Each segment, such as a code, global variables, or stacks, is identified by a segment number and offset. The segmentation hardware uses a segment table to map these two-dimensional addresses to one-dimensional physical memory. Each table entry contains the segment's base address and limit. If an offset exceeds the segment limit, an addressing error occurs. This approach allows logical memory organization, providing flexibility and ease of programming while the system manages physical memory allocation.

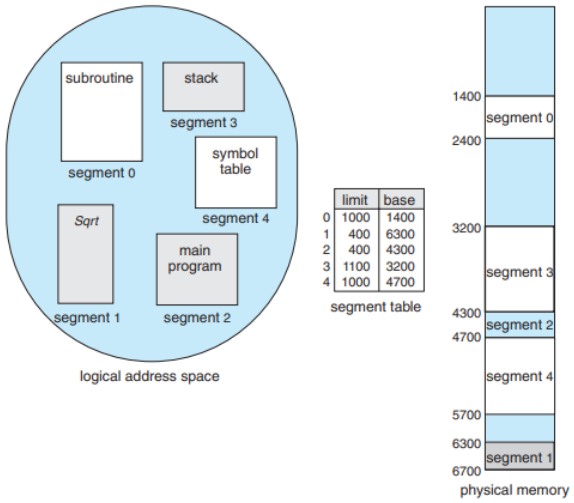


Figure 8.9 Example of segmentation.

Paging:
Segmentation and paging both allow noncontiguous physical address space for processes. Segmentation suffers from external fragmentation and requires compaction, while paging avoids these issues. Paging divides physical memory into fixed-size blocks called frames and logical memory into pages of the same size. Pages are loaded into available frames from a backing store with fixed-size blocks. The CPU generates addresses divided into page numbers and offsets, using a page table to map these to physical memory. This separation of logical and physical addresses allows efficient memory management and is widely used in modern operating systems.

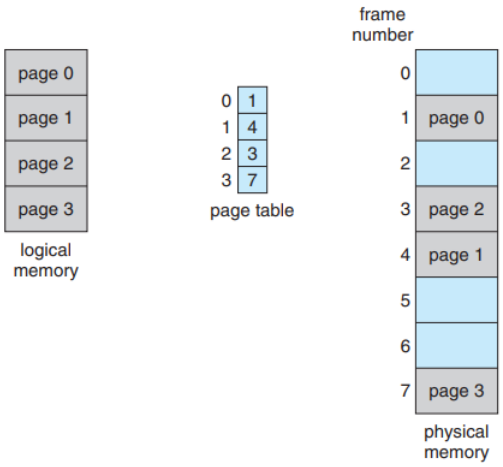


Figure 8.11 Paging model of logical and physical memory.

References:

1. Lecture Slide
2. Own Class Notes
3. Operating System Concepts 9th Edition by Abraham Silberschatz
4. ChatGPT for understanding some new topic
5. Grammarly for grammatical error check