Introduction to Process

Process

The process is an example of a computer program used. Contains the program code and its current function.

Program vs Process

Process	Program
1. The program contains a set of instructions designed to complete a task.	1. Process is an instance of an executing program.
2. Lifespan of the process is less than process	2. The lifespan of the program is longer
3. The process exists for a limited period as it gets terminated after the completion of a task.	3. Program exists in a single place and continues to exist until it is deleted.
4. Process is a dynamic entity.	4. Program is a static entity.
5. The process has a high resource requirement, it requires resources such as CPU, memory address, O / O during its lifetime.	5. The program has no resource requirement; it only requires memory space to store commands.

Table 1

Each process has a set of states that keeps on changing during its life cycle.

How does OS create a process?

The program or the application that should run is present in a file on a disk. So, the first thing that the OS does is it load the program and the static data into the memory. Static data are the data that is used to initialize the program.

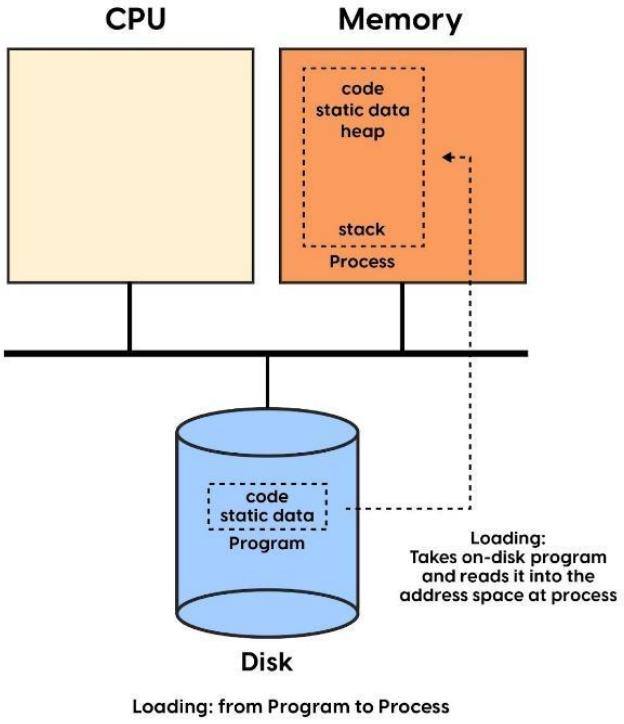

Figure 1

Once the program is loaded into the memory, the OS then allocates a runtime stack to the program. A stack is used for storing the local variables, function parameters, and return values for the program.

The OS will allocate some memory for the program's heap. Heap is basically the memory space that the program will use for dynamically allocated variables like link-list, hash tables, trees, and other data structures. Initially, the heap size allocated to the program by OS is small. However, as the used heap size grows, the program will request OS for more heap memory. OS will then increase the heap size for the program to satisfy its needs

The OS does some more initialization tasks related to Input/Output. E.g., in the case of Unix systems, OS opens three file descriptors for the program, standard input, standard output, and standard error. Using these descriptors, a program is able to read input and print output and error to the screen.

Once the program and static data is stored in the program, stack memory and heap memory are allocated to the program and the job related to I/O is done, the OS is ready to start the process.
The OS then jumps to the main () function for the program, the OS transfers the control of the CPU to the program. The program then starts execution and the process starts running.

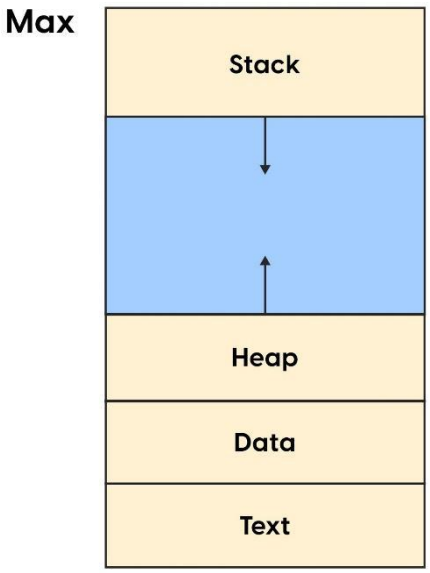

Figure 2

Based on the above diagram here are the definitions below:

1. **Text:** It consists of the compiled code that is read into the memory and was present on the non-volatile memory, i.e., disk
2. **Data:** Data is the global and static variables that are initialized prior to program execution
3. **Heap:** Heap is the part of the memory that is used for storing dynamically allocated data or variables.
4. **Stack:** The stack section is used for storing the return values, function parameters, and local variables.

Important Note: A heap and the stack are at the two ends of the program's allocated memory and they grow towards each other. If by any chance, they ever meet, the program gets a stack overflow error or if the program requests more memory, it will get an insufficient memory error.

Basics of Storage Devices

Introduction

To understand different states of the process, understanding different types of memory in our system is important. In our system, we have different types of memory present such as registers, caches, optical disk, etc. These memories are majorly split into two types: Primary memory and Secondary memory.

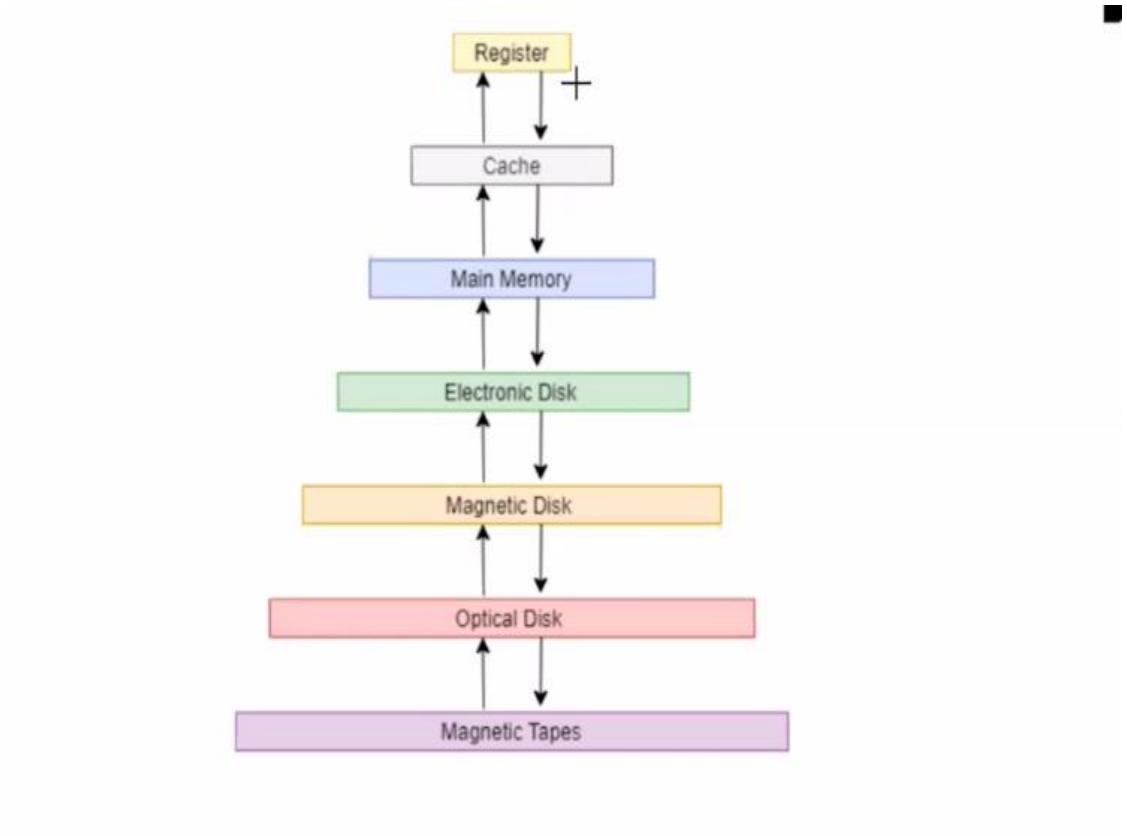

Figure 3

Primary Memory

The memory segment which can be directly accessed by a computer memory is called primary memory. It is also known as the volatile memory as the data is only there for runtime. Its access time is less than that of secondary memory. This type of memory has very less storage capacity compared to secondary memory. The following comes under the umbrella of primary memory:

Register: It is the smallest unit of storage as it stores only 0/1 (bits) but at the same time it has the fastest access time among all the memories. It has a very high cost and hence should only be used when speed is the utmost priority.

Cache: It is also a high-speed memory that is used to store the frequently used data and instructions to save a load of retrieval of the same information again and again. It is a less expensive way of storage than registers.

Main Memory: It is popularly known as RAM (Random Access Memory). Any process in the system which is going to be executed has to be loaded in RAM which is then processed by the CPU and it is done in accordance with the instructions of the program. It is probably slower than register and cache memory but still does the work. It is economical in cost and has very high storage capacity compared to registers and caches.

Secondary Memory

It is also known as external memory. It refers to the different storage media devices on which a computer can store its data and programs. It has a property that it can either be fixed or removed according to the need. A hard disk is fixed whereas a portable disk is removable. It is not directly accessed by the CPU as the data which is processed is first sent to primary memory (mostly RAM) and then to this memory for storage. The following comes under the secondary memory:

Electronic Disk: It is a data storage device that has a similar function as a hard drive but the only difference is that it uses flash memory.

Magnetic Disk: This type of disk is coated with a magnetic layer which is magnetized in different directions such as anticlockwise or clockwise. The head interprets the data stored at a specific location in 0/1 (binary) at reading whenever the head moves. Some examples are Hard disks and floppy disks.

Process States

1. **New State:** This is the state when the process is just created. It is the first state of a process.
2. **Ready State**: After creating the process, when the process is ready for its execution, it

goes into the ready state. In a ready state, the process is ready for its execution by the CPU but it is waiting for its turn to come. There can be more than one process in the ready state.

3. **Ready Suspended State:** There can be more than one process in the ready state but due to memory constraint, if the memory is full then some process from the ready state gets placed in the ready suspended state.
4. **Running State:** Amongst the process present in the ready state, the CPU chooses one process amongst them by using some CPU scheduling algorithm. The process will now be executed by the CPU and it is running.
5. **Waiting or Blocked State:** During the execution of the process, the process might require some I/O operation like writing on file or some more priority process might come. In these situations, the running process will have to go into the waiting or blocked state and the other process will come for its execution. So, the process is waiting for something in the waiting state.
6. **Waiting Suspended State:** When the waiting queue of the system becomes full, some of the processes will be sent to the waiting suspended state.
7. **Terminated State:** After the complete execution of the process, the process comes into the terminated state and the information related to this process is deleted.

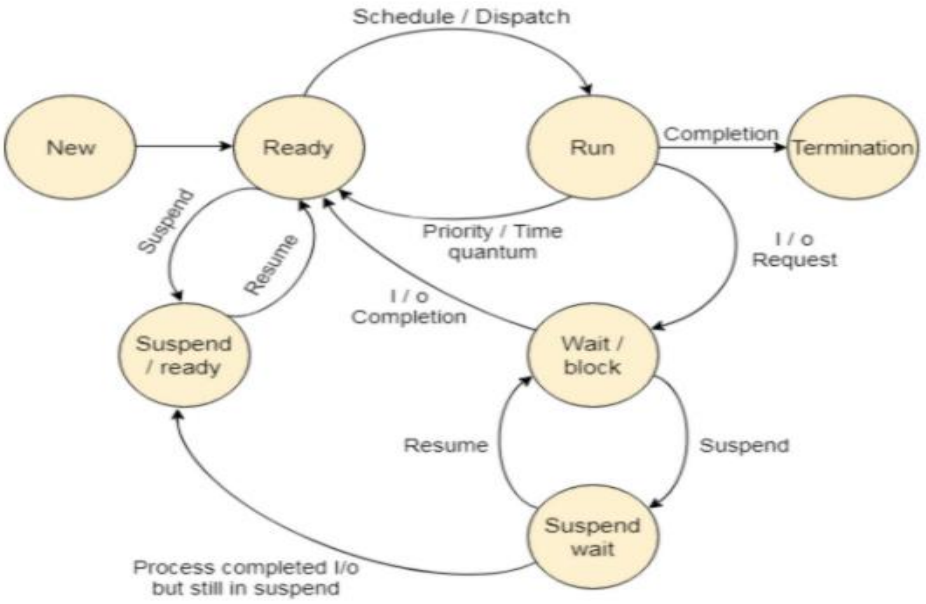

Figure 4

The information about a particular process is stored in a per-process data structure called Process Control Block.

Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB) also called a task control block. It contains many pieces of information associated with a specific process, including these:

1. **Process:** The state may be new, ready, running, and so on
2. **Program counter:** It indicates the address of the next instruction to be executed for this program.
3. **CPU registers:** These vary in number and type based on architecture. They include accumulators, stack pointers, general-purpose registers, etc.
4. **CPU scheduling:** This includes process priority, pointers to scheduling queues, and any scheduling parameters.
5. **Memory-management:** This includes the value of base and limit registers (protection) and page tables, segment tables depending on memory.
6. **Accounting:** It includes the amount of CPU and real-time used, account numbers, process numbers, etc.
7. **I/O status information:** It includes a list of I/O devices allocated to this process, a list of open files, etc

During the lifecycle of a process, it undergoes multiple state changes. These changes are facilitated by Process Schedulers. Process Schedulers use the following scheduling queues.

Scheduling Queue

Scheduling queues refer to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system. The operating system also maintains other queues such as device queues. A device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its device queue.

Queues are of three types:

1. **Job Queue:** As a process enters the system, it is put in a job queue that contains all the processes in the system.
2. **Ready queue:** The processes that are residing in the memory and are ready for execution are kept in the ready queue. The ready queue is implemented as a linked list of PCBs with a header containing pointers to the first and the last PCBs.
3. **Waiting Queue or Device queue:** The list of the processes waiting for a particular I/O device is called a device queue.

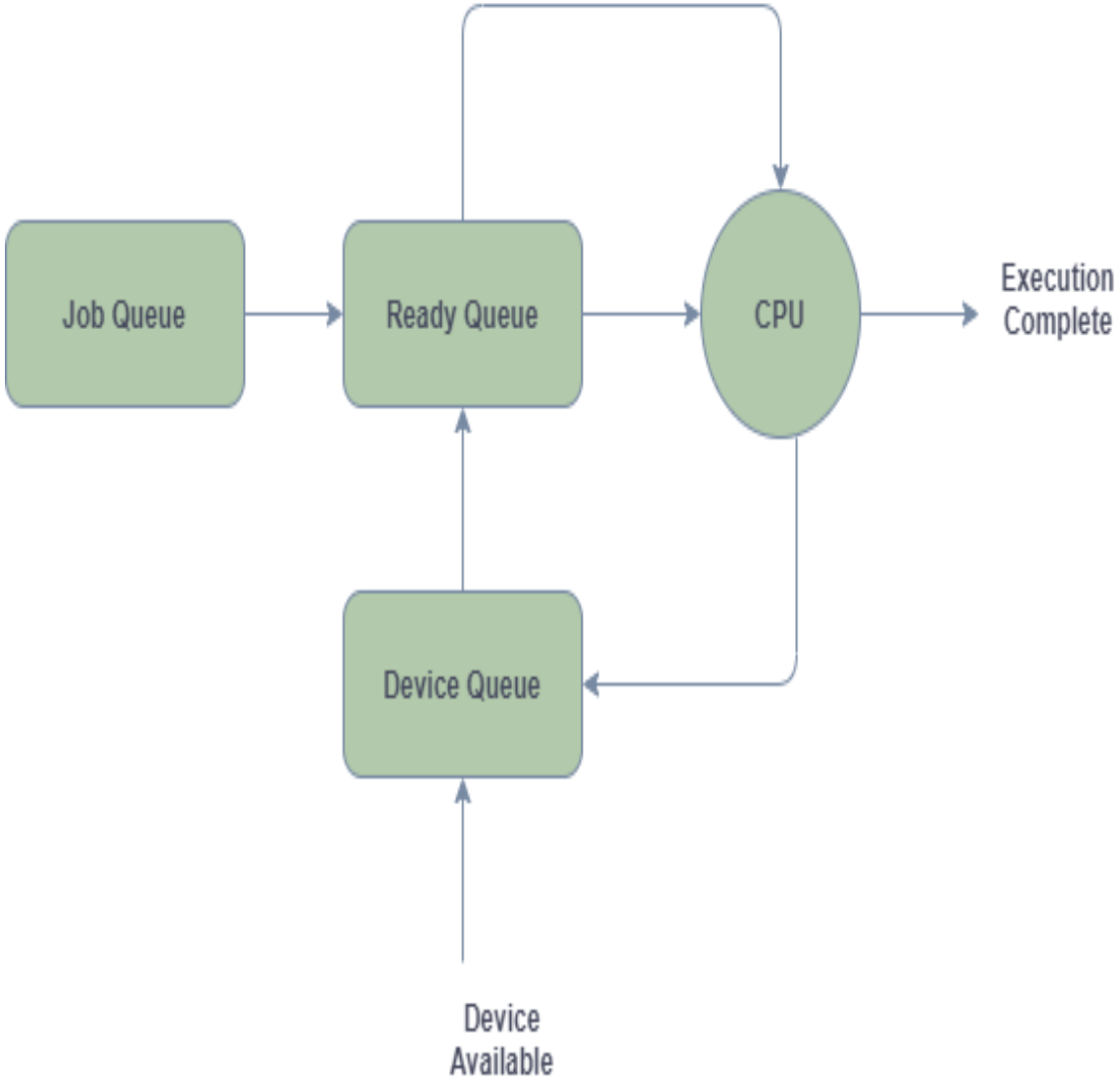

Figure 5

Types of Schedulers

There are three types of a scheduler:

1. **Long-Term Scheduler:** Long term scheduler is known as a job scheduler. It picks the process from the job queue and puts it into the ready queue. The ready queue is in the main memory. The long-term scheduler is used for managing the multitasking of the OS. When picking up the process from the job queue, this scheduler selects those processes that offer a mix of CPU-bound workload and an I/O bound workload. This means that if a process has to perform I/O intensive workload and low CPU-bound operations, then this scheduler will not select these processes. If the process performs I/O intensive work it will be in the blocked state and during that time CPU will be sitting idle. So, this will affect the OS's ability to multi-task. Therefore, this scheduler selects those processes that are a mix of CPU and I/O workload.

2. **Short Term Scheduler:** This scheduler is also known as a CPU scheduler because it works on scheduling processes that are in a ready state. It selects processes that are in the ready queue and dispatches them to the CPU for execution. This scheduler tries to boost the overall performance of the system based on a certain policy. The job of the short-term scheduler is very critical since if it selects a process that takes more time to run then another process will keep on waiting in the ready state for their turn to come. This condition is known as starving and it occurs if this scheduler makes a mistake in selecting the process.

3. **Medium-term scheduler:** Let's say a process is running. So, the state of the process is running. Now, let's say it has to perform some I/O operation. The process will chain to block or wait for the state and to allow another process to run. It will change the state to suspend the wait state and move the process from main memory to secondary memory. This movement from main memory to secondary memory is known as swapping or context switch and the medium-term schedule is responsible for this. So, the medium-term scheduler is responsible for suspending and resuming the process. This scheduler reduces the degree of multi-tasking.

Context Switching

Context switching is done to switch between processes. Switching the CPU to another process requires saving the current process's state and reloading the state of another process. States are saved into and reloaded from PCBs. Context-switch time is a pure overhead as the system does not do any useful work during a control switch. Context-switch time depends highly on the hardware. Context switching is faster on RISC processors with overlapped register windows. Apart from the process and its lifecycle, we learned about various operations such as process creation and process termination. We also learned about two types of processes: Orphan Processes and Zombie Processes

Orphan Process: The process whose parent gets terminated, while it is still running. In the case of Unix systems, this process is immediately adopted by the init process. Remember, the init process is the first process that runs on the Unix systems and has the PID

Zombie Process: Process which has completed its execution with the help of exit() system call, but still has an entry in the process table.

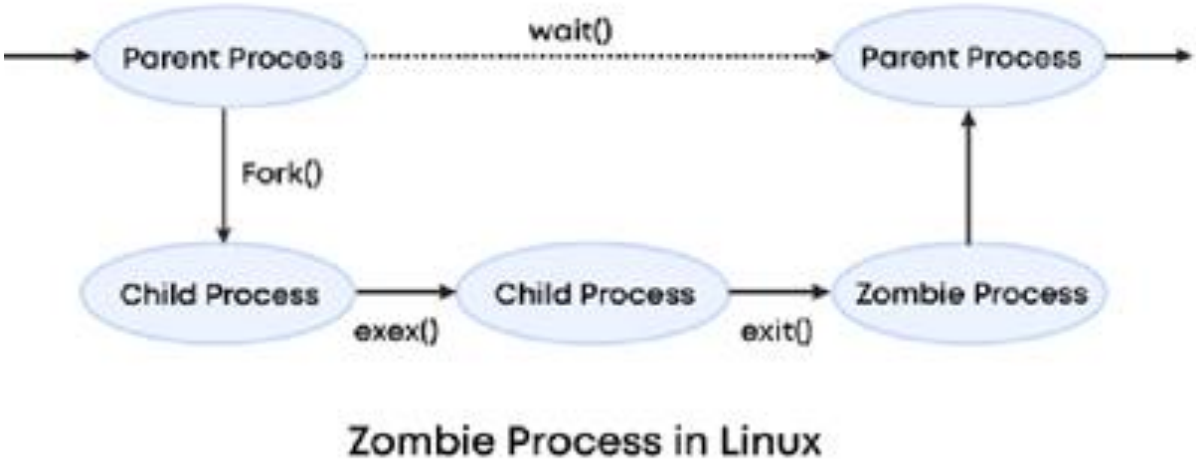

Figure 6