**Lecture 1: Intro**

OS is second most imp subject in CS after DSA.

Mainly it is used when we need to directly interact with the hardware/machine.

Topics like process synchronization and memory management are useful for software development.

OS is a software which takes control of the computer after machine is turned on power on tests are completed. It becomes the incharge of the machine.

Applications don’t directly interact with the machine. The interact with OS and OS with hardware.

Why do we need manager/incharge?

Abstraction: Without the OS we will have to write code for basic functionalities such as displaying text on screen, moving mouse cursor etc. OS contains the necessary codes for the basic functionalities and makes things easier. This also keeps the hardware abstract.

Resource Management: We have limited resources in our machine and using them effectively is difficult. OS manages the resources.

Protection: It also protects the hardware from applications and applications from each other.

Do we always need a manager?

Let us say we want to operate hardware like lifts and ovens. In such cases the chips are simple and only one or few simple tasks are to be done in such cases we don’t need OS.

Desktop OS: Windows, Linux, MacOS.

Mobile OS: Android, iOS.

There are more OS for printers, routers etc.

Three main services that OS provide:

Abstraction.

Resource Management.

Protection.

**Lec 02: Types of OS**

There can be many basis of classification.

Based on functionality provided by OS.

1. Single Tasking: MS-DOS, Only one process other than OS can exist in memory.
2. Multi Programming and Multi Tasking: Having multiple processes in RAM and assign them in smartly to CPU. Multiprogramming is general idea of managing multiple processes and multitasking is an extended version of multi-programming.
3. Multithreading:
4. Multiprocessing: For a system with multiple processors.

Thread: A thread is the smallest unit of execution/process that can be assigned to CPU. A process can be composed of single or multiple threads. Every OS uses concept of multi-threading nowadays.

Multi-user operating systems: Multiple users can use the same machine as different unique users.

**Lec 03: Thread vs Process**

Program gets loaded in RAM and then it is called a Process. A process is program in execution.

Pictorial representation of process with single thread. These are segments of a process.

|  |
| --- |
| Stack ↓ (Stack grows downwards) |
|  |
|  |
|  |
| Heap ↑ (Heap grows upward) |
| Text/Code |
| Data |

If a process is single threaded then it will have only one stack. For multi-threaded processes we have multiple stacks.

Pictorial representation of process with multiple threads.

|  |  |  |
| --- | --- | --- |
| Stack ↓ | Stack ↓ | Stack ↓ |
|  | | |
|  | | |
|  | | |
| Heap ↑ (Heap grows upward) | | |
| Text/Code | | |
| Data | | |

Multiple threads have multiple stacks but same Heap, data and code.

Concurrent and parallel have different meanings with reference to process execution.

More about threads:

1. Faster to create and terminate.
2. Share same address space.
3. Easier to communicate.
4. Context switching is easier.
5. Lightweight.

**Lec 04: Multithreading Intro**

* Multithreading vs Multitasking
* Some real world examples
* Advantages and Disadvantages

Multitasking: Listening to music and browsing. (Multiple tasks are being done)

Multithreading: Downloading and browsing. (Multiple things are begin done within a process)

Real world examples of multithreading:

MS Word: Typing, saving, formatting is done together using multithreading.

IDEs: Error checking is done while the text is formatted.

Advantages of multithreading:

1. Parallelism and improved performance
2. More responsiveness
3. Better resource utilization

Threads are also called light weight processes.

Disadvantages of Multithreading:

1. Difficulty in writing, testing and debugging code.
2. Can lead to deadlock and race conditions. (mainly when variables are shared, like in language JAVA)

R2

R1

Deadlock: T1 thread holds R1 resource and is waiting for R2. Meanwhile T2 holds R2 and is waiting for R1. Until T1 releases R1, T2 cannot be finished and similarly for T1.

**Lec 05: User Threads vs Kernel Threads**

User managed threads: The threads created by a process and the kernel is not aware about the threads and the process manages the threads.

Kernel managed threads: Managed by kernel and kernel is aware of everything going on.

|  |  |  |
| --- | --- | --- |
|  | User Managed Threads | Kernel Managed Threads |
| Management | In user space | In kernel space |
| Context Switching | Fast | Slow |
| Blocking | One thread can block all other threads | A thread can block itself only. |
| Multicore or Multiprocessor | Cannot take advantage of multicore systems. Only concurrent execution on single processor. | Takes full advantage of multicore systems. |
| Creation/Termination | Fast | Slow |

Usually every process have both kind of threads.

One to one: One user thread is mapped to only one kernel thread. On other user thread is mapped to this kernel thread. This is most common. This resembles with purely kernel managed threading system.

Many to one: Multiple user threads are mapped to one kernel thread. This resembles with purely user managed threading system.

Many to many: Multiple user threads are connected to multiple kernel threads. Very uncommon.

**Lec 06: Intro to process**

Program: File that resides on hard disk which can be executed.

Process: A program in execution.

The binary file (.exe) is loaded into RAM and then run. While it runs it is called a process.

|  |
| --- |
| Stack ↓ (Stack grows downwards) (for function calls, local variables) |
|  |
|  |
|  |
| Heap ↑ (Heap grows upward) (head for dynamically allocated memory) |
| Text/Code (Instructions to be executed) |
| Data (static and global variables) |

**Lec 07: Process States**

Single tasking systems (MS – DOS)

New Process-> Goes in Memory -> Finished/Terminated

In such systems processor remains idle for long.

Multiprogramming System: Systems which can have multiple processes in RAM other than OS simultaneously. They have 5 state model.

Running

New

(

Finished

Ready

Waiting

New State: We just double clicked an exe file. Process control info is created. But the exe file is not yet loaded in RAM, it is still in HD. It is also said that process is not created yet.

Ready State: exe file is partially or completely in RAM. Now it is said that process has been created and ready to be picked by the processor.

When the required part in in RAM. The dispatcher takes it to CPU and the process is said to be dispatched.

If a higher priority process comes or time-out happens the process is sent back to ready state.

If the process requests I/O word it is sent to waiting state from running state.

Once the I/O is done, the process is again sent to ready state and then dispatched.

If the process gets finished or aborted, it is sent to finished state.

The seven-state model:

Two more states come out of waiting state.

New

(

Ready

Running

Finished

Waiting

Suspend

Resume

Suspend

Resume

Suspend/Ready

When all the current processes are in waiting state, CPU is idle. In such cases, one of the processes is sent back to HD to pick a new process from HD.

Suspend/Block

While waiting the process is in RAM but on suspending the process is in HD.

Process can be directed to suspended state while waiting or while in ready state.

While in any suspended state the process is in HD but suspended state means that process has been through ready state for at least once and has not completed yet.

There is a dedicated place for suspended processes. Swap partition/space is available in Linux for these. In windows, a folder, page file stores these processes. (Things might have changed over time.)

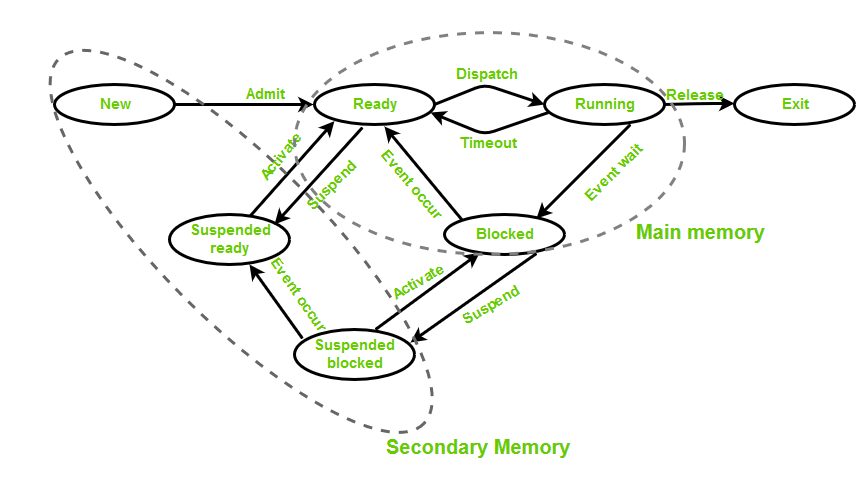


Diagram from GeeksForGeeks

Note: CPU interacts with RAM only. It doesn’t interact with HD. CPU doesn’t pick up processes. It is responsibility of OS to assign process to CPU.

**Lec 08: Process State Block**

PCB is a data structure. It is the most central data structure of OS. It is the most secure DS so that id doesn’t get corrupted.

It stores info regarding process state. Some of the imp info is:

1. Process Id (Integer)
2. Process state
3. CPU Register
4. Accounts info (how much CPU time has been consumed etc.)
5. I/O Info
6. CPU scheduling info
7. Memory Info (memory blocks allocated to process etc.)

One imp CPU register is program counter which tells the next instruction to be executed.

**Lec 09: Process Schedulers**

I/O Bound Processes:

CPU Bound Processes:

1. Long Term Scheduler: Brings process from HD to RAM. Controls the degree of multi programming. Ideally it should bring a good mix of I/O bound and CPU bound processes. Decisions made by it occur on large gap that is why it is named so.
2. Short Term Scheduler: Moves the process from ready to running state. It assigns processor one of the picked processes. It has to make frequent decisions that is why it is named so. It also calls dispatcher. Dispatcher does the process control switch.
3. Medium Term Scheduler: It manages the suspending, blocking and resuming of process. It can be said that it moves processes from HD to RAM and vice versa.

**Lec 10: Background for Scheduling Algos**

The most crucial part is which process should be picked from ready state.

Most of the algos we study are for short term scheduler or for the jobs in ready Q.

* Different Queues: Ready Q, Job Q, I/O Q (each I/O device has its own Q).
* Short Term Scheduler and Dispatcher: STS picks one of the processes from ready Q and dispatcher comes into light. Once a process is picked, Dispatcher does the context switch, mode switch and tells from where a program should begin (a program might come from suspended state so it should not run from beginning).

When does STS picks a process?

1. When some process moves from running to waiting state.
2. When some other process moves from running to ready state.

Pre-emptive cases

1. When a new/existing process moves to ready state.
2. When a process is aborted or terminated.

Time related terms in Scheduling Algos:

1. Arrival Time (Point):
2. Completion Time (Point):
3. Burst Time (Period):
4. Turn Around Time (Period):
5. Waiting Time (Period):
6. Response Time (Period):­

Goals of Scheduling Algos:

1. Max CPU utilization
2. Max throughput (throughput is number of jobs finished in unit time.)
3. Min Turn Around Time
4. Min Waiting Time
5. Min Response Time
6. Fair CPU Allocation (No Starvation)

Starvation: If a process has been waiting for a long time to get picked and executed then it is said to be starving.

Almost every time we care much about averages of the time.

GANTT Chart: Representation of CPU utilization by process with time stamps.