**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) |
| Data |
| Code |

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) | | |
| Data | | |
| Code | | |

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.