Administrative details

Textbook: Operating System Concepts — *Silberschatz, Galvin, Gagne*

Other references:

1. [Vahalia] Unix Internals The New Frontiers — *Uresh Vahalia* (Pearson Education Asia/LPE)
2. [Bach] The Design of the UNIX Operating System — *M.J. Bach* (Prentice Hall)
3. [ULK] Understanding the Linux Kernel — *Bovet, Cesati* (O'Reilly)
4. [APUE] Advanced Programming in the UNIX Environment — *W. Richard Stevens* (Addison-Wesley), 1992.
5. [UNP] UNIX Network Programming — *W. Richard Stevens* (Prentice Hall), 1990.

Weightage: Assignments 20% Mid-sem 30% End-sem 50%

Slides: Available from `http://www.isical.ac.in/~mandar/courses.html`

What is an OS?

Definition

Software that manages a computer's hardware resources for its users and their applications

What is an OS?

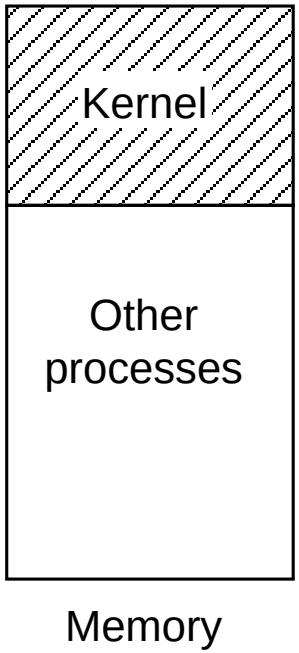
a library of functions + set of programs

Definition

Software that manages a computer's hardware resources for its users and their applications

Components of OS

- Kernel: core library that provides functions for basic operations (e.g., process creation / destruction) + interface to hardware via ***API*** (Application Programming Interface)
- Processes / programs
  - system processes – daemons/servers (`httpd`, `lpd`, `sendmail`, etc.)
  - user processes – shell, editor, compiler, utilities



Why do we need an OS?

- *Convenience*
  - mediates access to hardware by providing convenient abstractions (not easy to use hardware directly)
  - provides environment + services needed to run user programs in a convenient way
- *Resource sharing* between multiple users / processes
- *Protection/security:* prevent different users / processes from interfering with each other
- *Communication:* coordinate operation of different processes

Things to manage

- CPU (processes)
- RAM (memory management)
- Hard discs (file systems)
- Keyboard, monitor (I/O devices)

Call stack: review

- *Activation record (AR):* block of memory used to store information pertaining to a function (*local variables*, *parameters*, *return value*, etc.)
- AR allocated / deallocated when function is called / returns
  - variables created when function is called; destroyed when function returns
- Function calls behave in *last in first out* manner
  $\Rightarrow$ use *stack* to keep track of ARs
- Information that needs to be shared between calling function and called function (*parameters*, *return value*, *return address*) stored at the boundary between the two ARs
- Stack Pointer register (*SP*) points to AR at top of stack