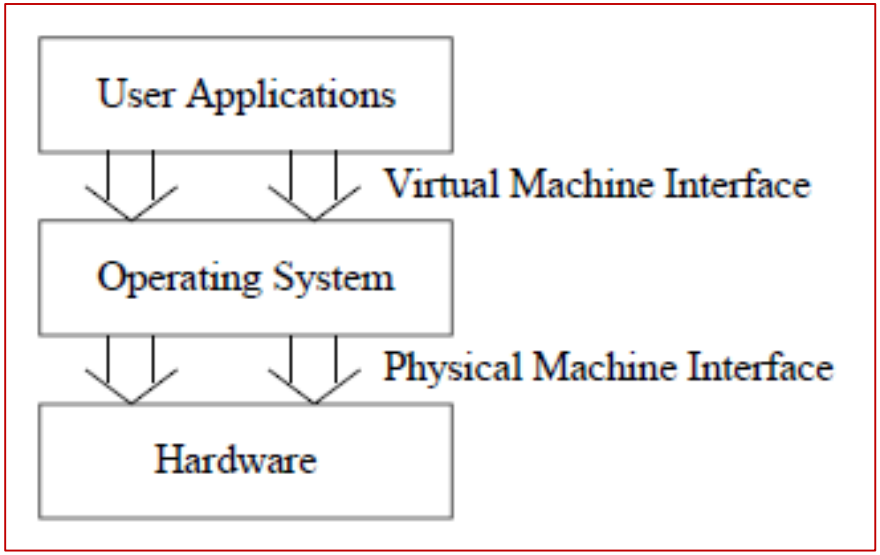
CSCI Lecture 1 Supplement

What is an Operating System



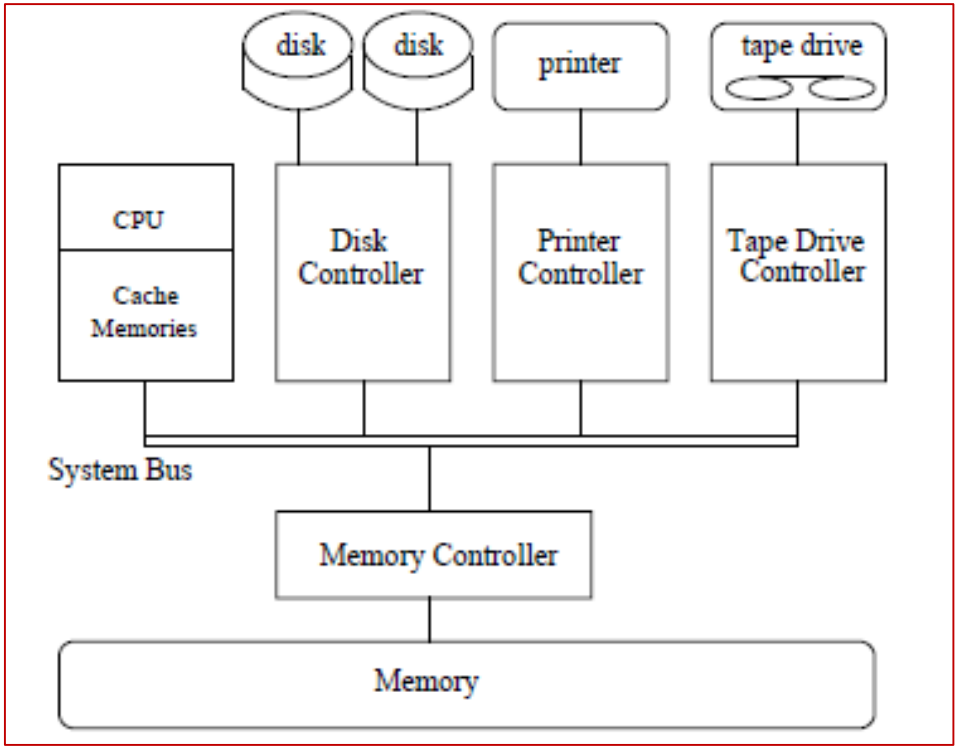
- **OS as a juggler –** providing the illusion of a dedicated machine with infinite memory and CPU
- **OS as government –** protecting users from each other, allocating resources efficiently and fairly, and providing secure and safe communication
- **OS as complex system –** keeping OS design and implementation as simple as possible is the key to getting the OS to work
- **OS as history teacher –** learning from past to predict the future, i.e. OS design tradeoffs change with technology

Computer Architecture & Process

- What the OS can do is dictated in part by the architecture
  - The architectural features can greatly simplify or complicate the OS

- Process is a unit of execution
  - How are processes represented in the OS?
  - What are possible execution states and how does the system move from one state to another?

Generic Computer Architecture

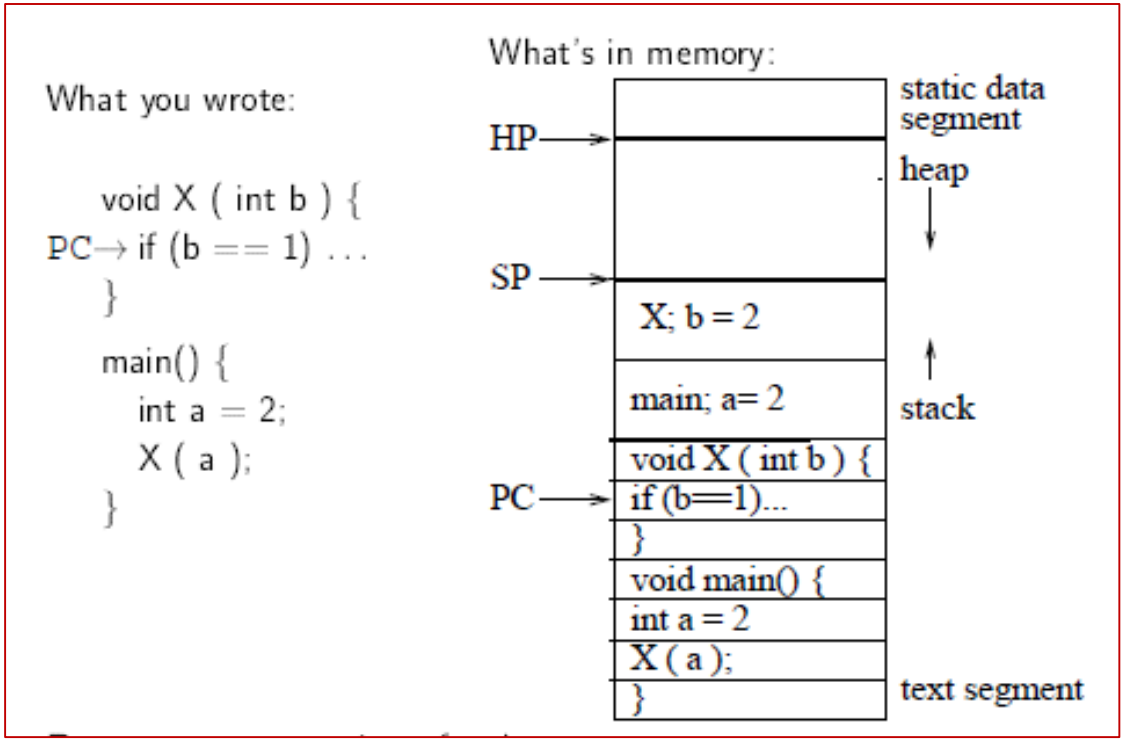


- CPU – the processor that performs the actual computation
- I/O devices – terminal, disks, video board, printer, etc.
- Memory – RAM containing data and program used by the CPU
- System bus – the communication medium between the CPU, memory, and peripherals

What's in a Process

- A process is the dynamic execution context of an executing program
  - Several processes may run the same program, but each is a distinct process with its own stat (e.g., vi)
  - A sequential process is a program execution that yields a sequential, one instruction at a time, ordering of the instructions

Example Process State in Memory



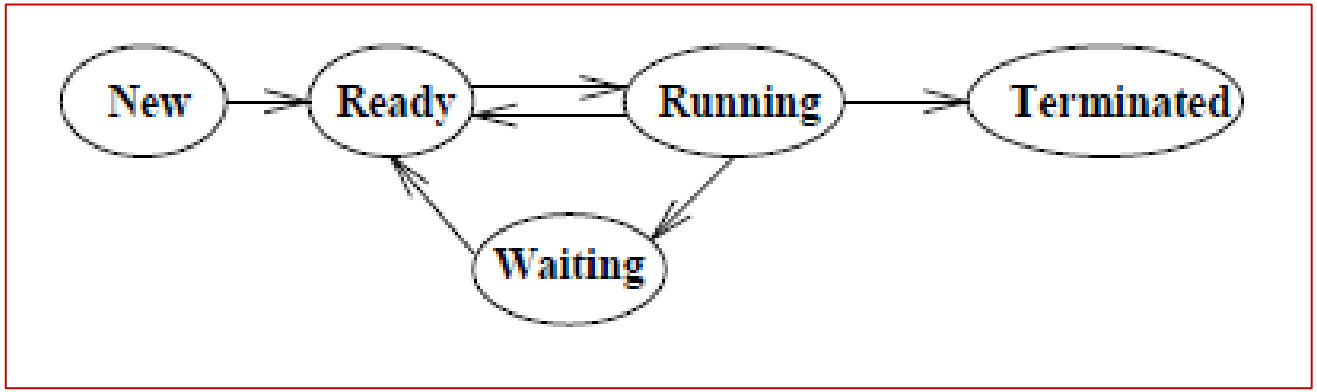
- Process state consists of at least:
  - The code for running the program
  - The program counter (PC) indicating the next instruction
  - An execution stack with the program's call chain (the stack), the stack pointer (SP)
  - The static data for the running program
  - Space for dynamic data (the heap), the heap pointer (HP)
  - Values of CPU registers
  - A set of OS resources in use (e.g., open files)
  - Process execution state (e.g., ready, running, etc.)

Process Execution State

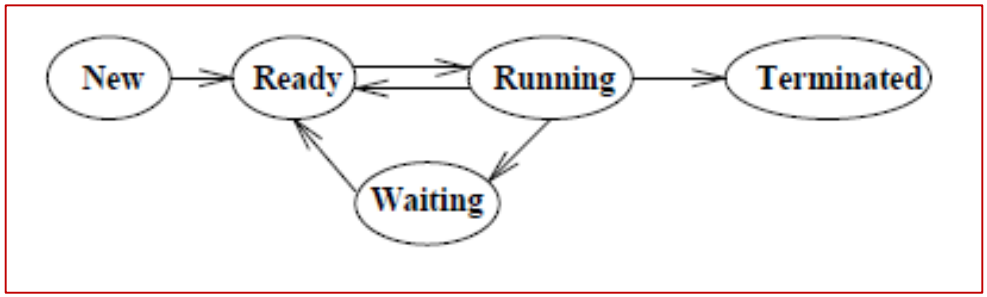
- Each process has an execution state which indicates what it is currently doing

New	the OS is setting up the process state
Ready	ready to run, but waiting for the CPU
Running	executing instructions on the CPU
Waiting	waiting for an event to complete (e.g., I/O)
Terminated	the OS is destroying this process

- As the program executes, it moves from state to state, as a result of the program actions (e.g., system calls), OS actions (scheduling), and external actions (interrupts)



Process Execution State



Example:

```
void main() {
  printf("Hello World");
}
```

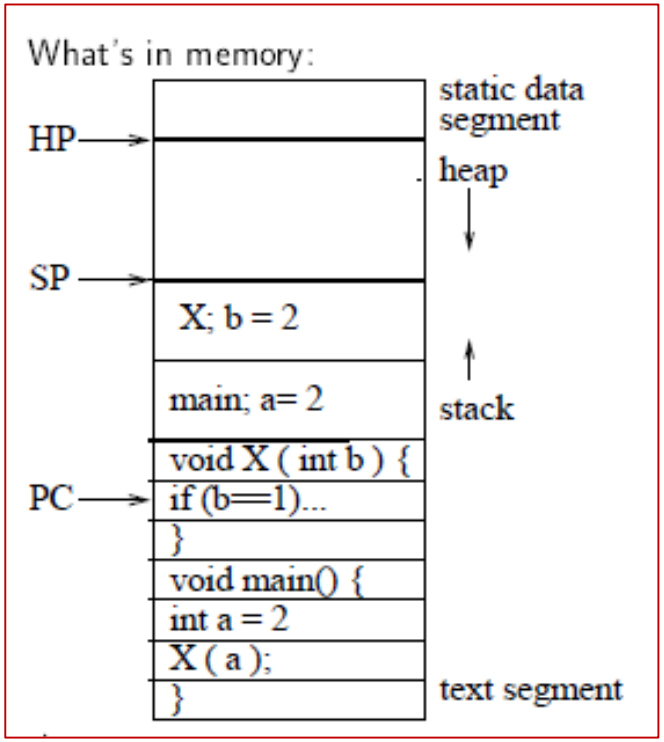
state sequence

new
ready
running
waiting for I/O
ready
running
terminated

- Since multiple processes are active at any one time, the OS manages them by placing all of the processes in the same state in a state queue (e.g., the ready queue)

OS Process Data Structures – Process Control Block

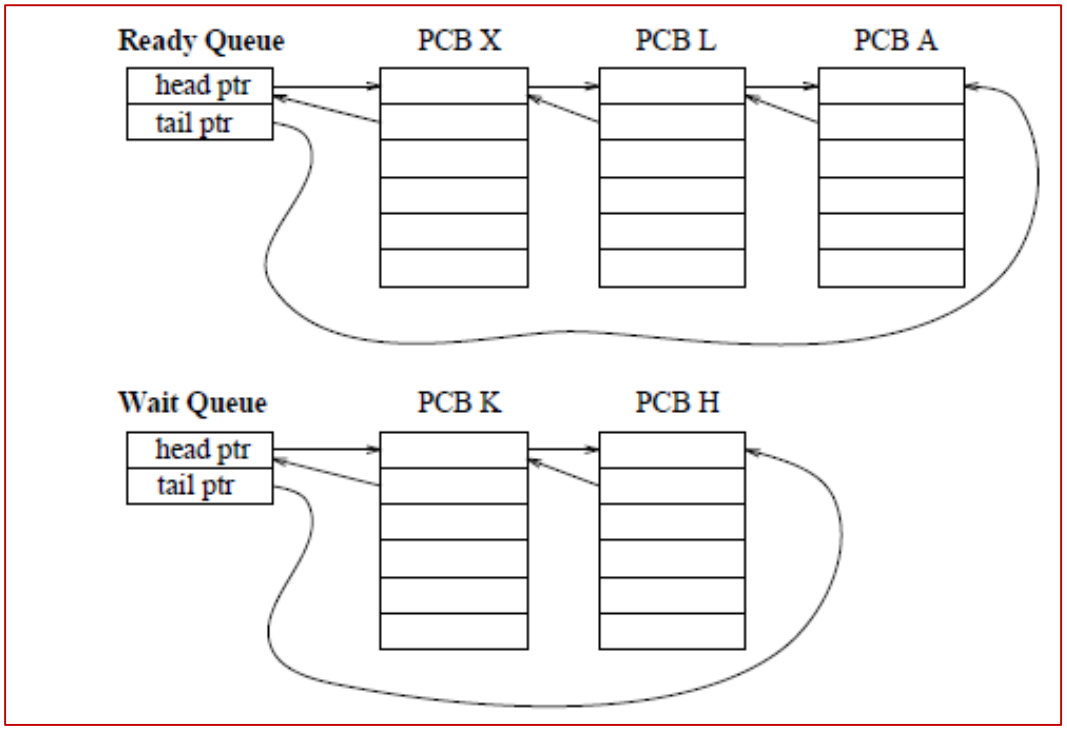
- The OS uses a Process Control Block (PCB), its own dynamic data structure, to keep track of all the processes (this data structure is in addition to the process state in memory). The PCB represents the execution state and location of each process when it is not executing (e.g., when it is waiting)
- The PCB contains
  - Process state (running, waiting, etc.)
  - Process number
  - Program Counter (PC)
  - Stack Pointer
  - General Purpose Registers
  - Memory Management Information (HP, etc.)
  - Username of owner
  - List of open files
  - Queue pointers for state queues (e.g., the waiting queue)
  - Scheduling information (e.g., priority)
  - I/O status
  - …



- When a process is created, the OS allocates and initializes a new PCB, and then places the PCB on a state queue
- When a process terminates, the OS deallocates the PCB

OS Process State Queues

- The OS maintains the PCBs of all the processes in the system by using state queues
  - The OS places the PCBs of all the processes in the same execution state in the same queue
  - When the OS changes the state of a process, the PCB is unlinked from its current queue and moved to its new state queue
  - The OS can use different policies to manage each queue
  - Each I/O device has its own wait queue



PCBs and Hardware State

- Starting and stopping processes is called a context switch, and is a relatively expensive operation
  - When the OS is ready to start executing a waiting process, it loads the hardware registers (PC, SP, and other registers) from the values stored in that process PCB
  - While a process is running, the CPU modifies the Program Counter (PC), Stack Pointer (SP), registers, etc…
  - When the OS stops a process, it saves the current values of the registers, (PC, SP, etc.) into the PCB for that process
  - This process of switching the CPU from one process to another (stopping one and starting the next) is the context switch
  - Time sharing system may to 100 to 1000 context switches a second (the cost to do the context switch and the time between process switches are closely related)

Creating a Process

- Example
  - When you log in to a machine running Linux, you create a shell process
  - Every command you type into the shell is a child of your shell process and is an implicit fork and exec pair
  - For example, you type “vi”, the OS “forks” a new process and then “exec” (executes) “vi”
  - If you type an “&” after the command, Linux will run the process in parallel with your shell, otherwise, your next shell command must wait until the first one completes

Summary

- A process is the unit of execution
- Processes are represented as Process Control Blocks in the OS that encapsulate a program state, memory management information, scheduling information, I/O status, etc.
- At any time, a process is either New, Ready, Waiting, Running, or Terminated
- On a uniprocessor, there is at most one Running process at a time
- The program currently executing on the CPU is changed from performing a context switch