

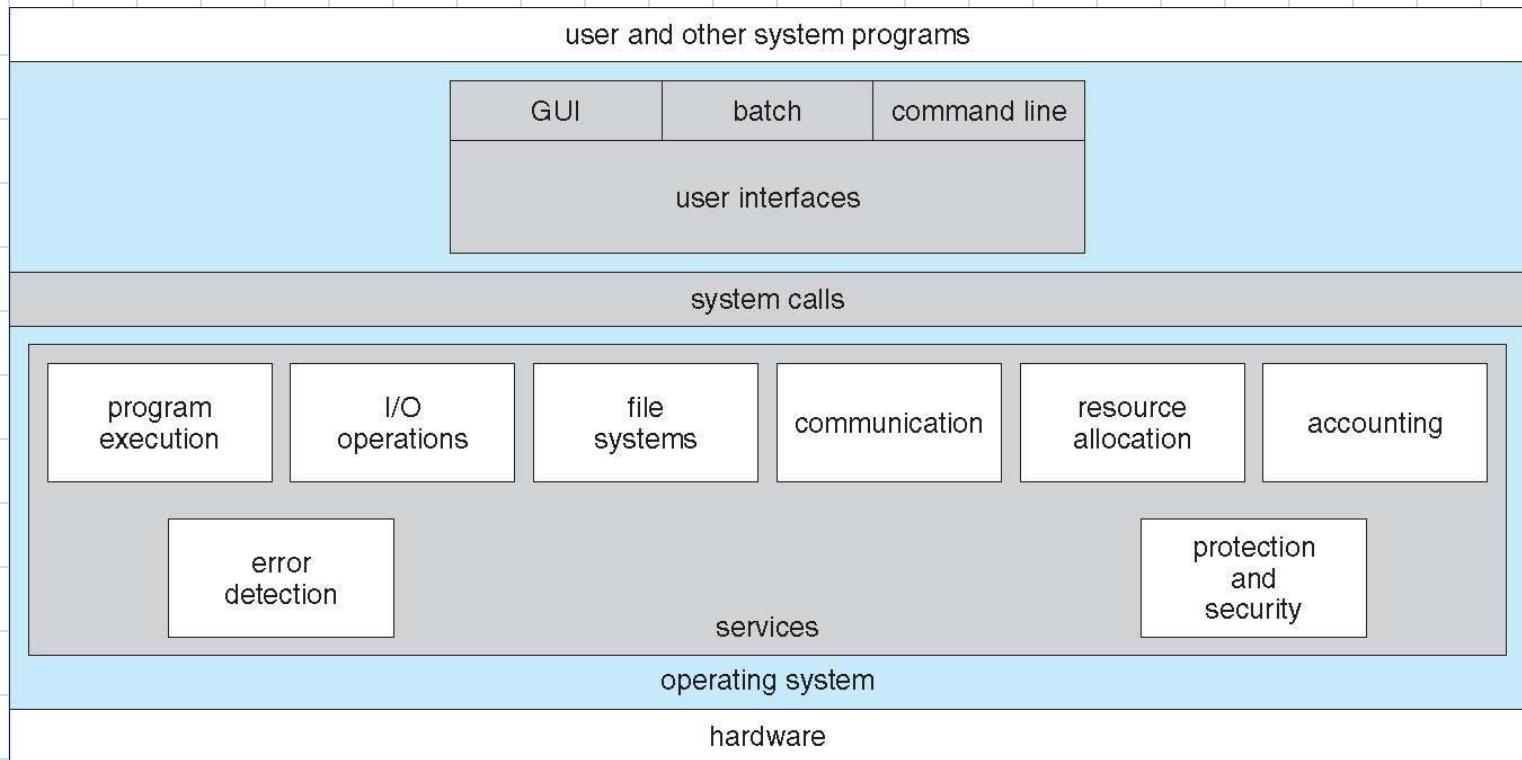
The Process Model



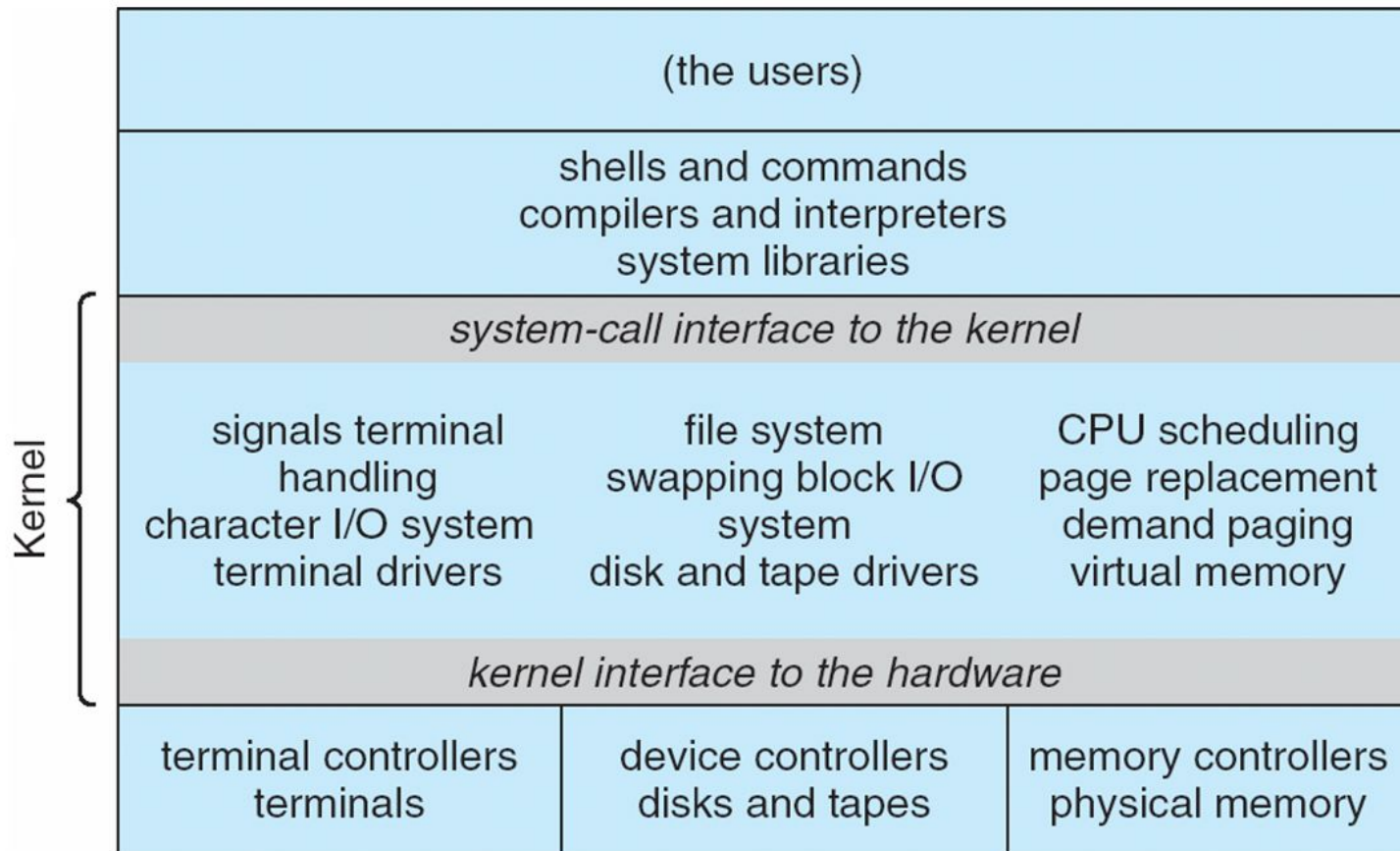
Topics

- Review system call
- Introduce the process model
 - To introduce the notion of a process -- a program in execution, which forms the basis of all computation
 - To describe the various features of processes, including scheduling, creation and termination, and communication
 - To describe communication in client-server systems

A View of Operating System Services



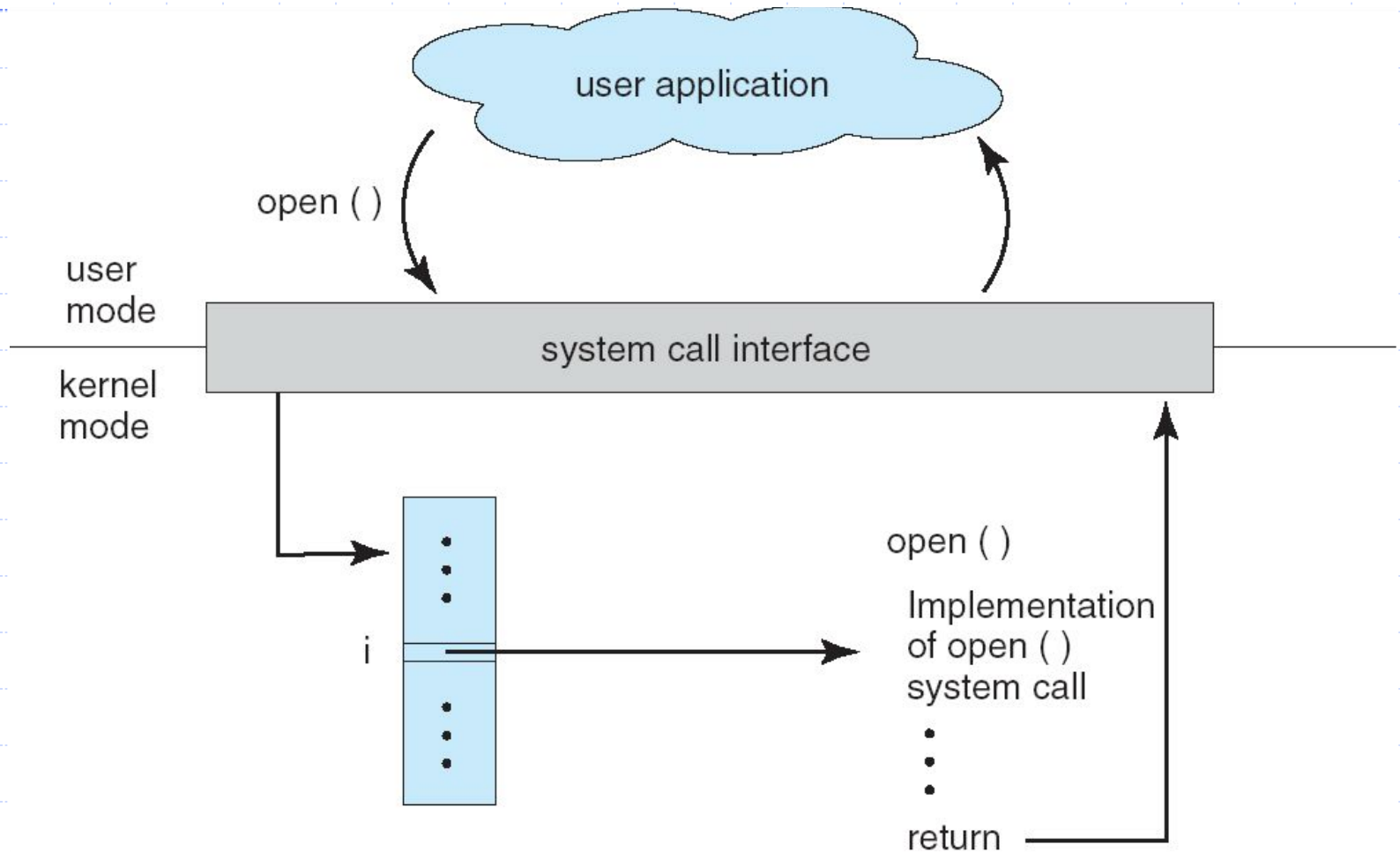
Traditional UNIX System Structure



System Call Implementation

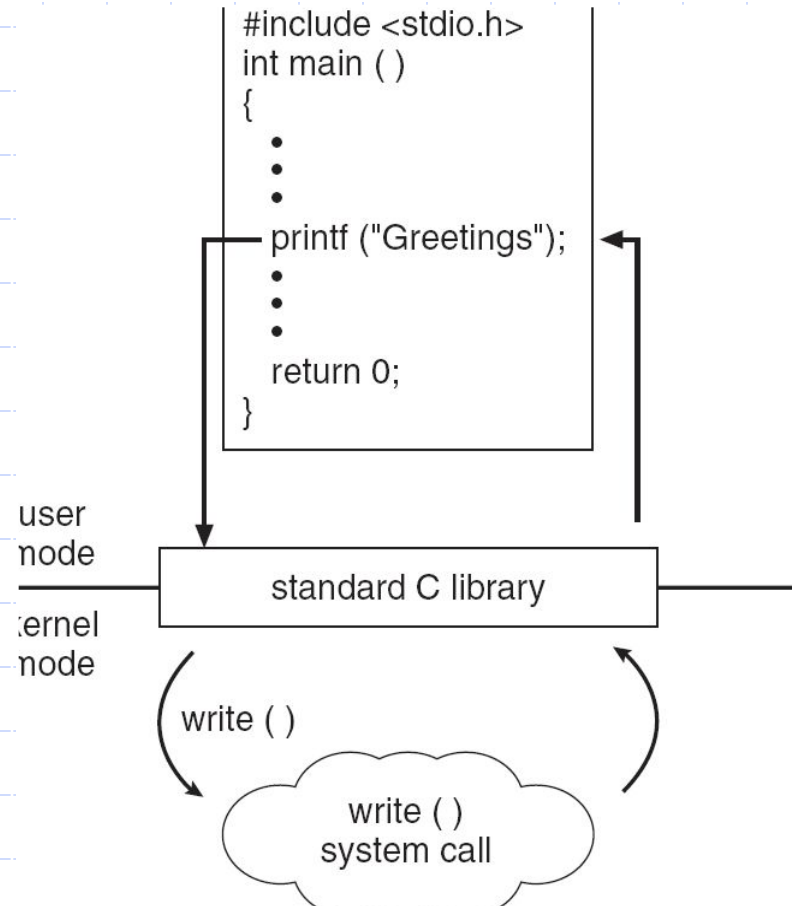
- Typically, a number associated with each system call
 - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most details of OS interface hidden from programmer by API
 - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



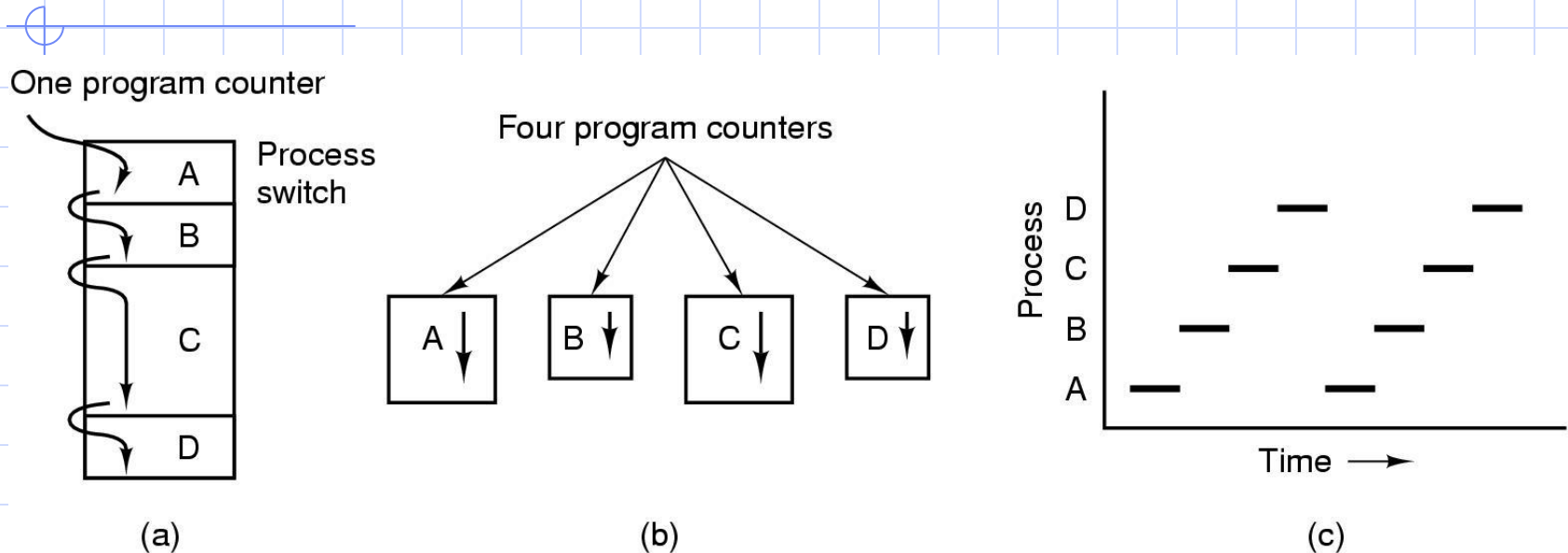
Standard C Library Example

- C program invoking printf() library call, which calls write() system call



Processes

The Process Model

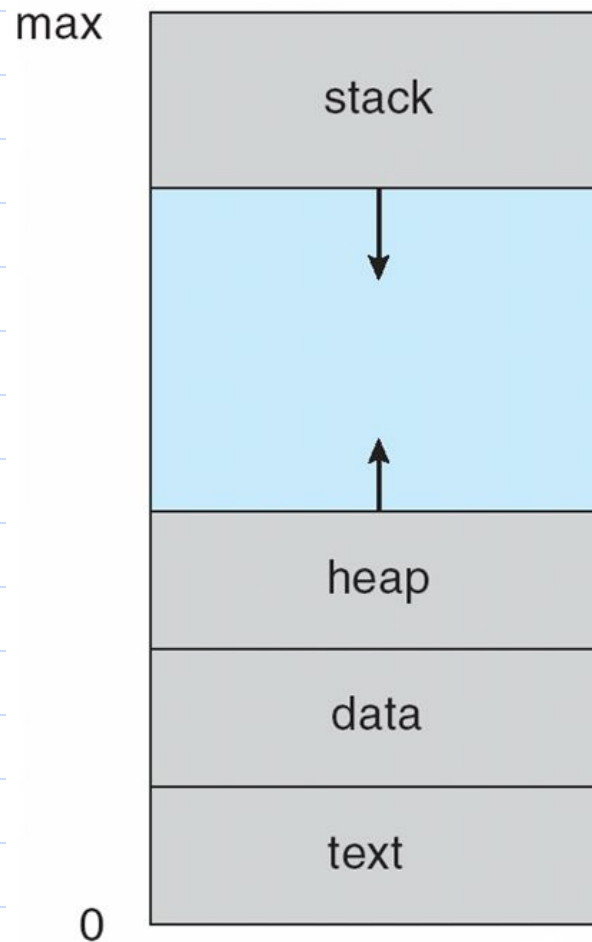


- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant

What is a process?

- A process is simply a program in execution: an instance of a program execution.
- Unit of work individually schedulable by an operating system.
- A process includes:
 - program counter
 - stack
 - data section
- OS keeps track of all the active processes and allocates system resources to them according to policies devised to meet design performance objectives.
- To meet process requirements OS must maintain many data structures efficiently.
- The process abstraction is a fundamental OS means for management of concurrent program execution. Example: instances of process co-existing.

Process in Memory



Process creation

- Four common events that lead to a process creation are:
 - 1) When a new batch-job is presented for execution.
 - 2) When an interactive user logs in / system initialization.
 - 3) When OS needs to perform an operation (usually IO) on behalf of a user process, concurrently with that process.
 - 4) To exploit parallelism an user process can spawn a number of processes.

Termination of a process

- Normal completion, time limit exceeded, memory unavailable
- Bounds violation, protection error, arithmetic error, invalid instruction
- IO failure, Operator intervention, parent termination, parent request, killed by another process
- A number of other conditions are possible.
- **Segmentation fault** : usually happens when you try write/read into/from a non-existent array/structure/object component. Or access a pointer to a dynamic data before creating it. (new etc.)
- **Bus error**: Related to function call and return. You have messed up the stack where the return address or parameters are stored.

Process control

- Process creation in unix is by means of the system call `fork()`.
- OS in response to a `fork()` call:
 - Allocate slot in the process table for new process.
 - Assigns unique pid to the new process..
 - Makes a copy of the process image, except for the shared memory.
 - both child and parent are executing the same code following `fork()`
 - Move child process to Ready queue.
 - **it returns pid of the child to the parent, and a zero value to the child.**

Process control (contd.)

- All the above are done in the kernel mode in the process context. When the kernel completes these it does one of the following as a part of the dispatcher:
 - Stay in the parent process. Control returns to the user mode at the point of the fork call of the parent.
 - Transfer control to the child process. The child process begins executing at the same point in the code as the parent, at the return from the fork call.
 - Transfer control another process leaving both parent and child in the Ready state.

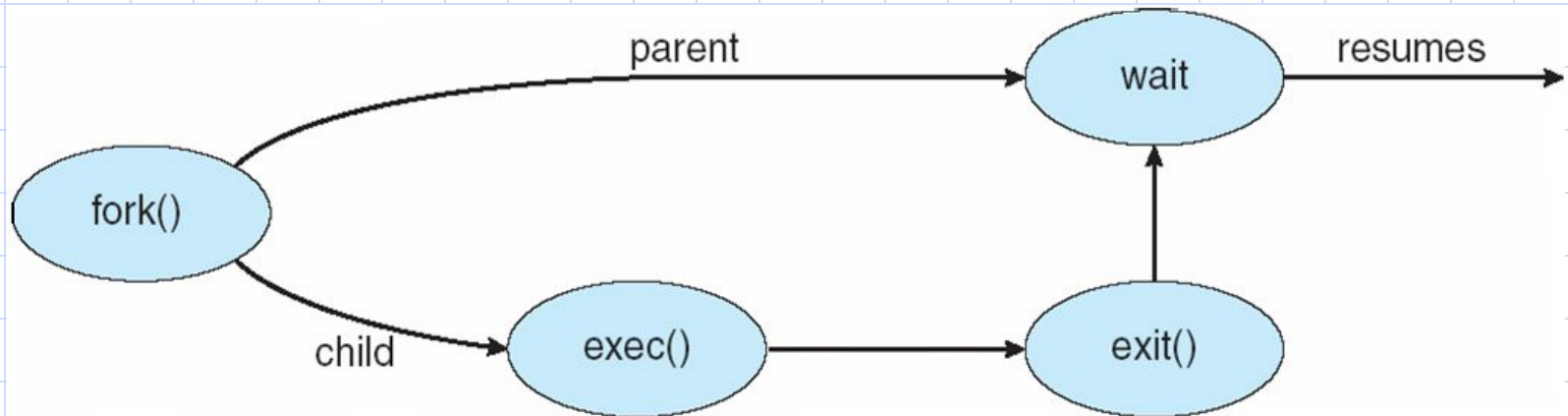
Process Creation (contd.)

- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via **a process identifier (pid)**
- Resource sharing
 - Parent and children share all resources
 - Children share subset of parent's resources
 - Parent and child share no resources
- Execution
 - Parent and children execute concurrently
 - Parent waits until children terminate

Process Creation (Contd.)

- Address space
 - Child duplicate of parent
 - Child has a program loaded into it
- UNIX examples
 - **fork** system call creates new process
 - **exec** system call used after a **fork** to replace the process' memory space with a new program

Process Creation (contd.)



C Program Forking Separate Process

```
int main() {  
    int retVal;  
    /* fork another process */  
    retVal = fork();  
    if (retVal < 0) { /* error occurred */  
        fprintf(stderr, "Fork Failed");  
        exit(-1);  
    }  
    else if (retVal == 0) { /* child process */  
        execlp("/bin/ls", "ls", NULL);  
    }  
    else { /* parent process */  
        /* parent will wait for the child to  
        complete */  
        wait (NULL);  
        printf ("Child Complete");  
        exit(0);  
    } }  
}
```

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
 - Output data from child to parent (via **wait**)
 - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required
 - If parent is exiting
 - Some operating system do not allow child to continue if its parent terminates
 - All children terminated - **cascading termination**

fork and exec

- Child process may choose to execute some other program than the parent by using exec call.
- Exec overlays a new program on the existing process.
- Child will not return to the old program unless exec fails. This is an important point to remember.
- Why does fork need to clone?
- Why do we need to separate fork and exec?
- Why can't we have a single call that fork a new program?

Example

```
if (( result = fork()) == 0 ) {  
    // child code  
    if (execv ("new program",...) < 0)  
        perror ("execv failed ");  
    exit(1);  
}  
else if (result < 0 ) perror ("fork"); ...}  
/* parent code */
```

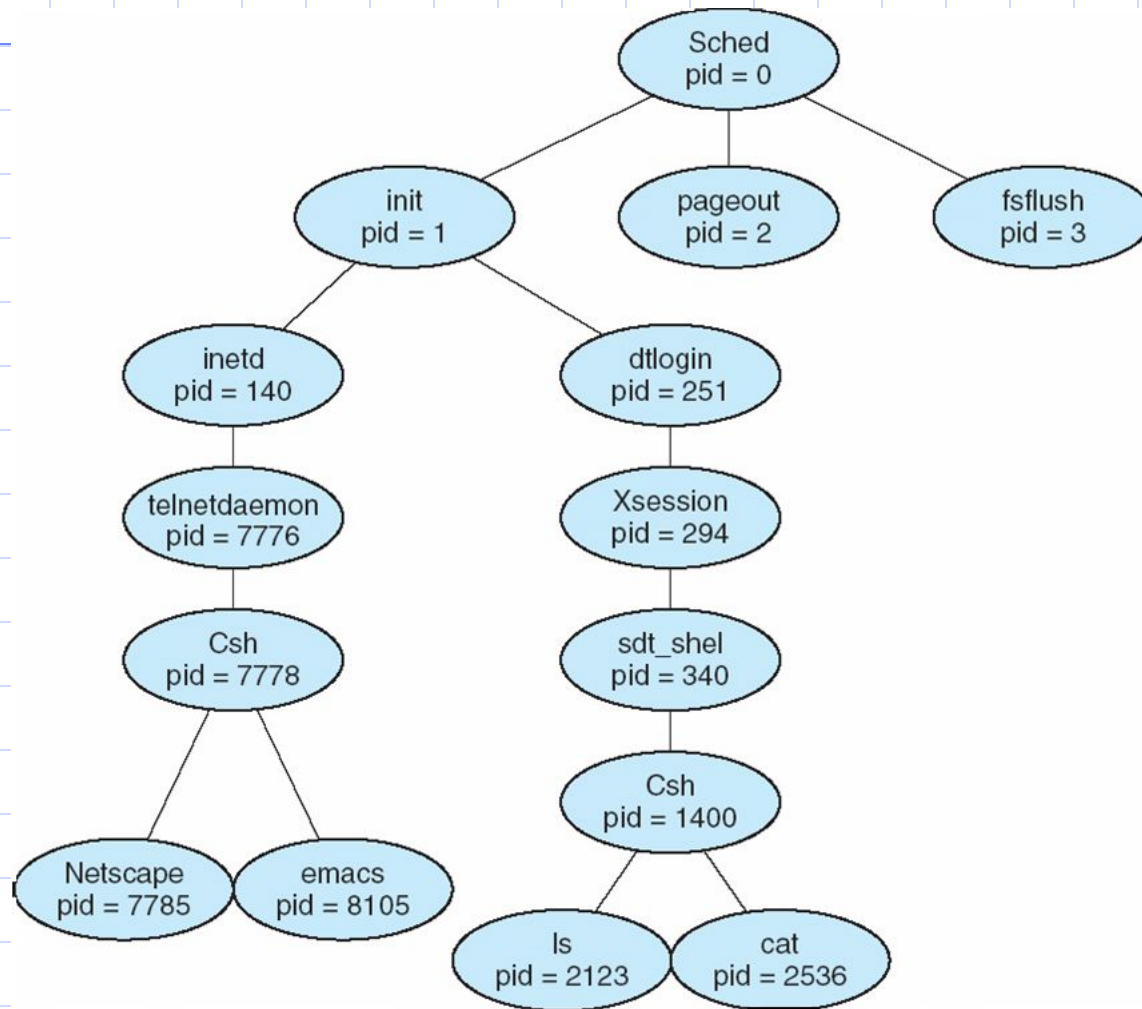
Versions of exec

- Many versions of exec are offered by C library: exece, execve, execvp, execl, execl, execlp
- We will look at these and methods to synchronize among various processes (wait, signal, exit etc.).

Process Hierarchies

- Parent creates a child process, child processes can create its own process
- Forms a hierarchy
 - UNIX calls this a "process group"
- Windows has no concept of process hierarchy
 - all processes are created equal

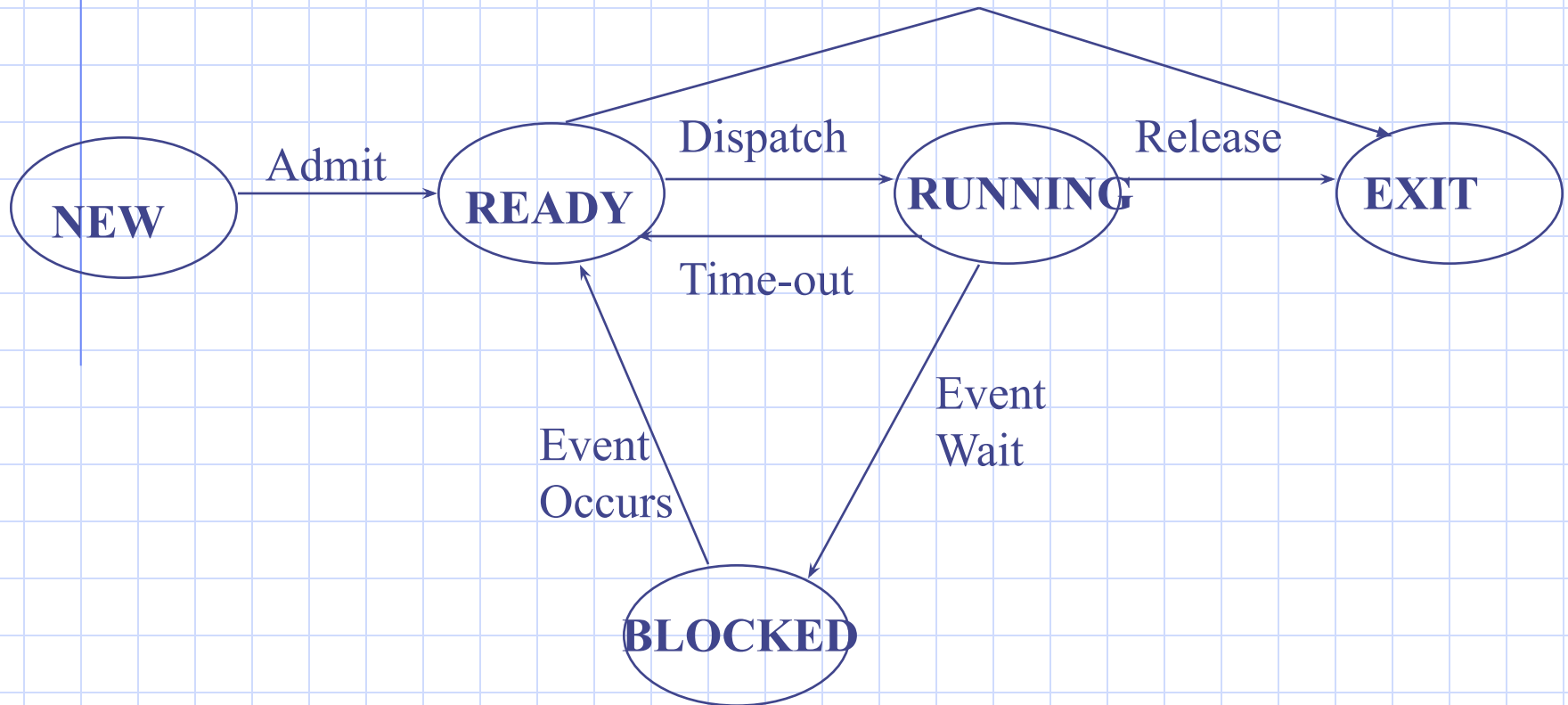
A tree of processes on a typical Unix system



A five-state process model

- Five states: New, Ready, Running, Blocked, Exit
- **New** : A process has been created but has not yet been admitted to the pool of executable processes.
- **Ready** : Processes that are prepared to run if given an opportunity. That is, they are not waiting on anything except the CPU availability.
- **Running**: The process that is currently being executed. (Assume single processor for simplicity.)
- **Blocked** : A process that cannot execute until a specified event such as an IO completion occurs.
- **Exit**: A process that has been released by OS either after normal termination or after abnormal termination (error).

State Transition Diagram (1)



Think of the conditions under which state transitions may take place.

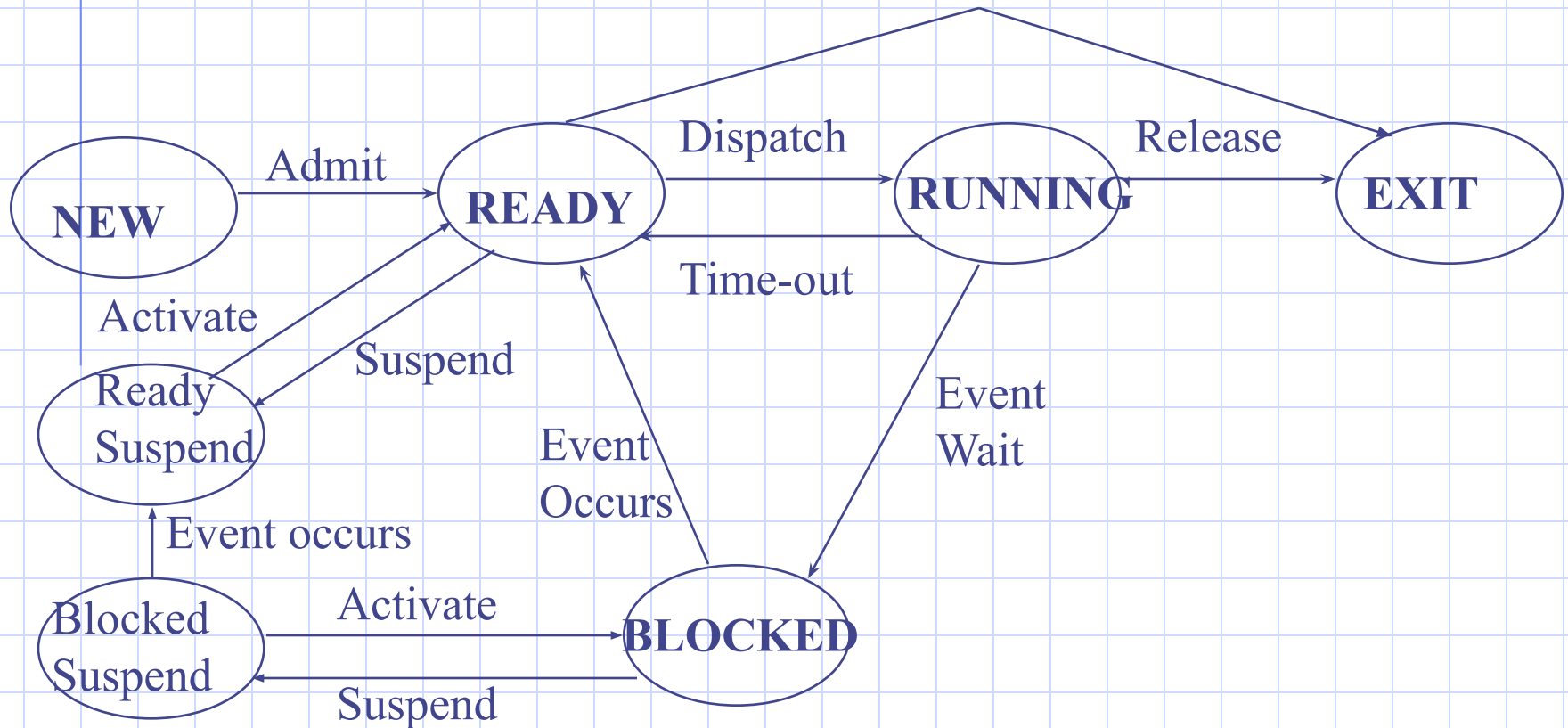
Process suspension

- Many OS are built around (Ready, Running, Blocked) states. But there is one more state that may aid in the operation of an OS - **suspended** state.
- When none of the processes occupying the main memory is in a Ready state, OS swaps one of the blocked processes out onto to the Suspend queue.
- When a Suspended process is ready to run it moves into "Ready, Suspend" queue. Thus we have two more state: Blocked_Suspend, Ready_Suspend.

Process suspension (contd.)

- Blocked_suspend : The process is in the secondary memory and awaiting an event.
- Ready_suspend : The process is in the secondary memory but is available for execution as soon as it is loaded into the main memory.
- State transition diagram on the next slide.
- Observe on what condition does a state transition take place? What are the possible state transitions?

State Transition Diagram (2)



Think of the conditions under which state transitions may take place.

Implementation of Processes

Process management

Registers
Program counter
Program status word
Stack pointer
Process state
Priority
Scheduling parameters
Process ID
Parent process
Process group
Signals
Time when process started
CPU time used
Children's CPU time
Time of next alarm

Memory management

Pointer to text segment
Pointer to data segment
Pointer to stack segment

File management

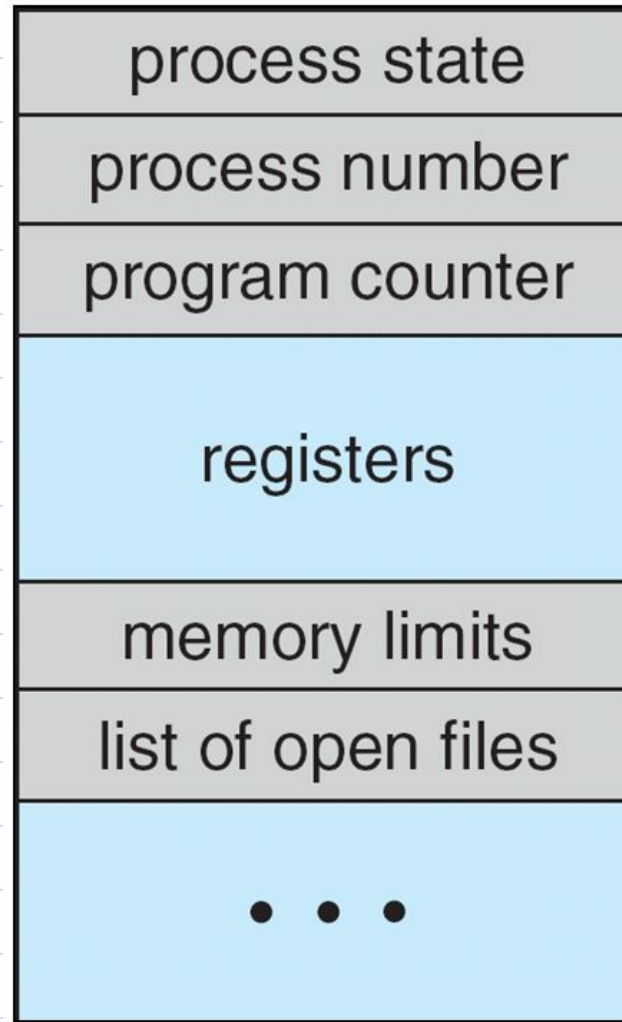
Root directory
Working directory
File descriptors
User ID
Group ID

Process Control Block (PCB)

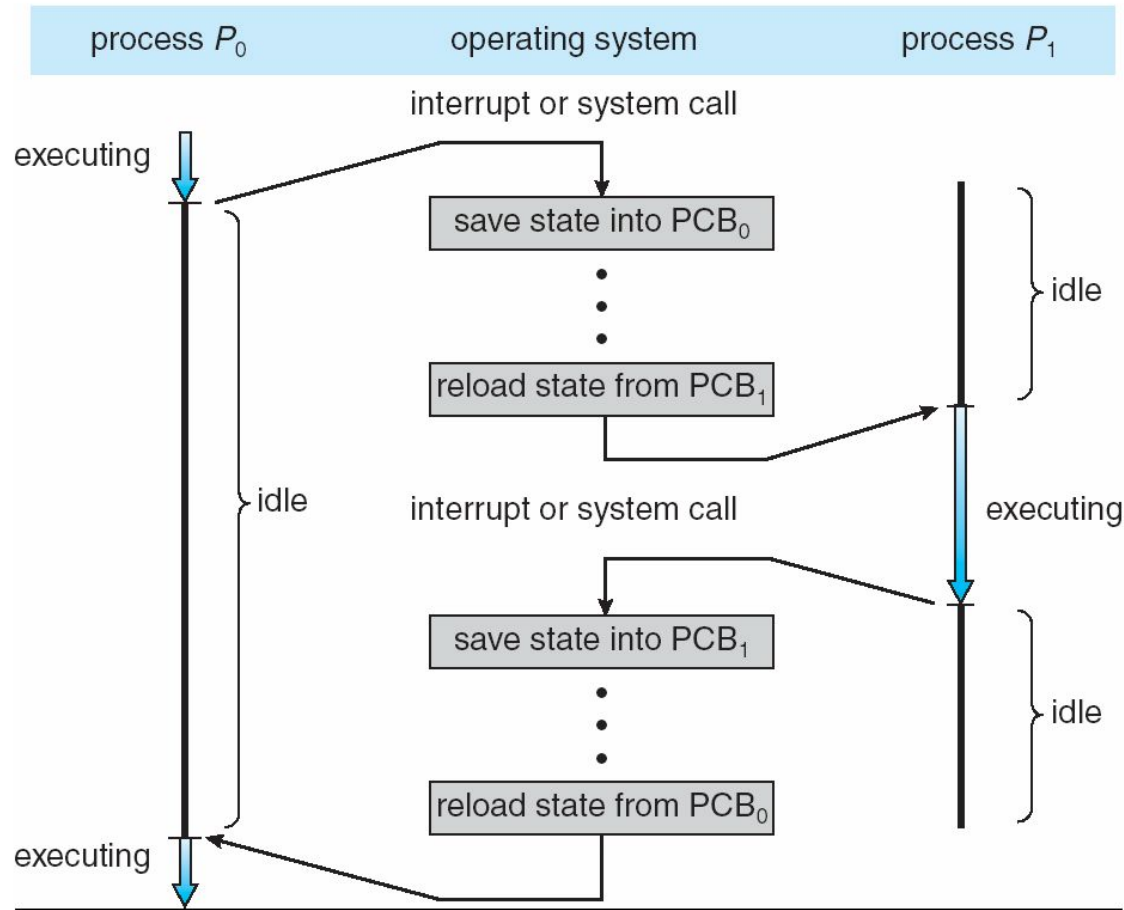
Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



CPU Switch From Process to Process



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Summary

- A process is a unit of work for the Operating System.
- Implementation of the process model deals with process description structures and process control methods.
- Process management is the of the operating system requiring a range of functionality from interrupt handling to IO management.
- All the concepts discussed will be illustrated in the project 1.