LEADING UNIVERSITY

estd. 2001

CSE-3211, Operating System

Chapter 3: Processes

Md Saidur Rahman Kohinoor
Lecturer, Dept. of Computer Science
Leading University – Sylhet
kohinoor_cse@lus.ac.bd

Objectives

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

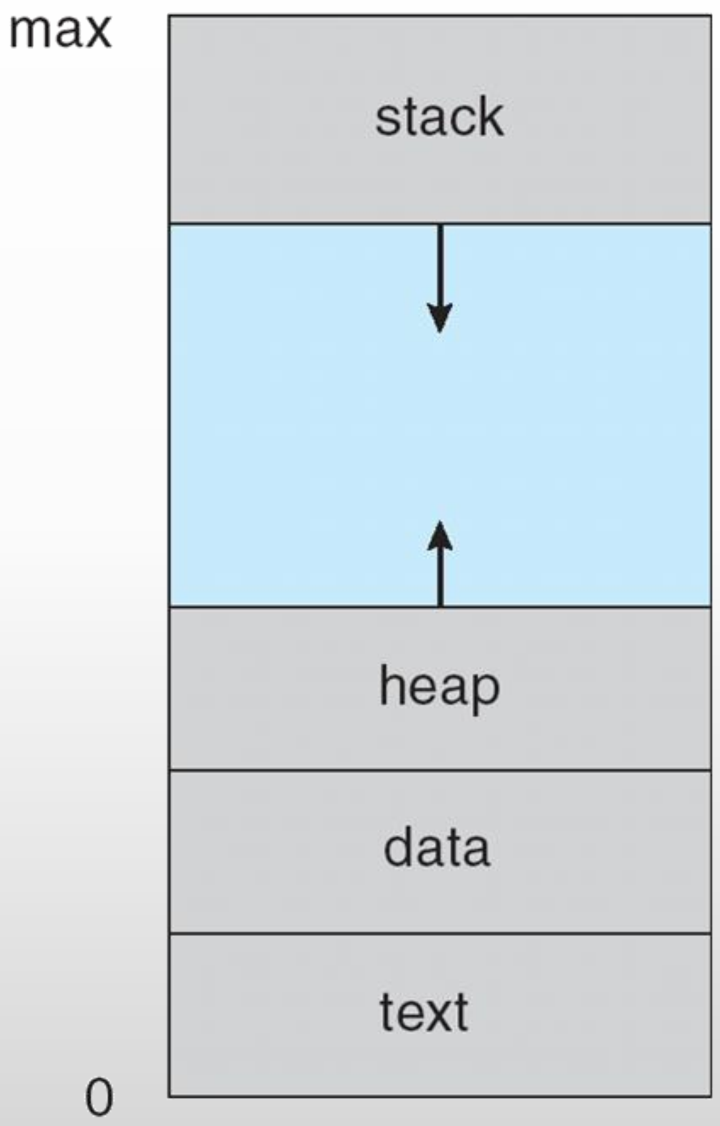
To explore interprocess communication using shared memory and mes- sage passing

To describe communication in client-server systems

Process Concept

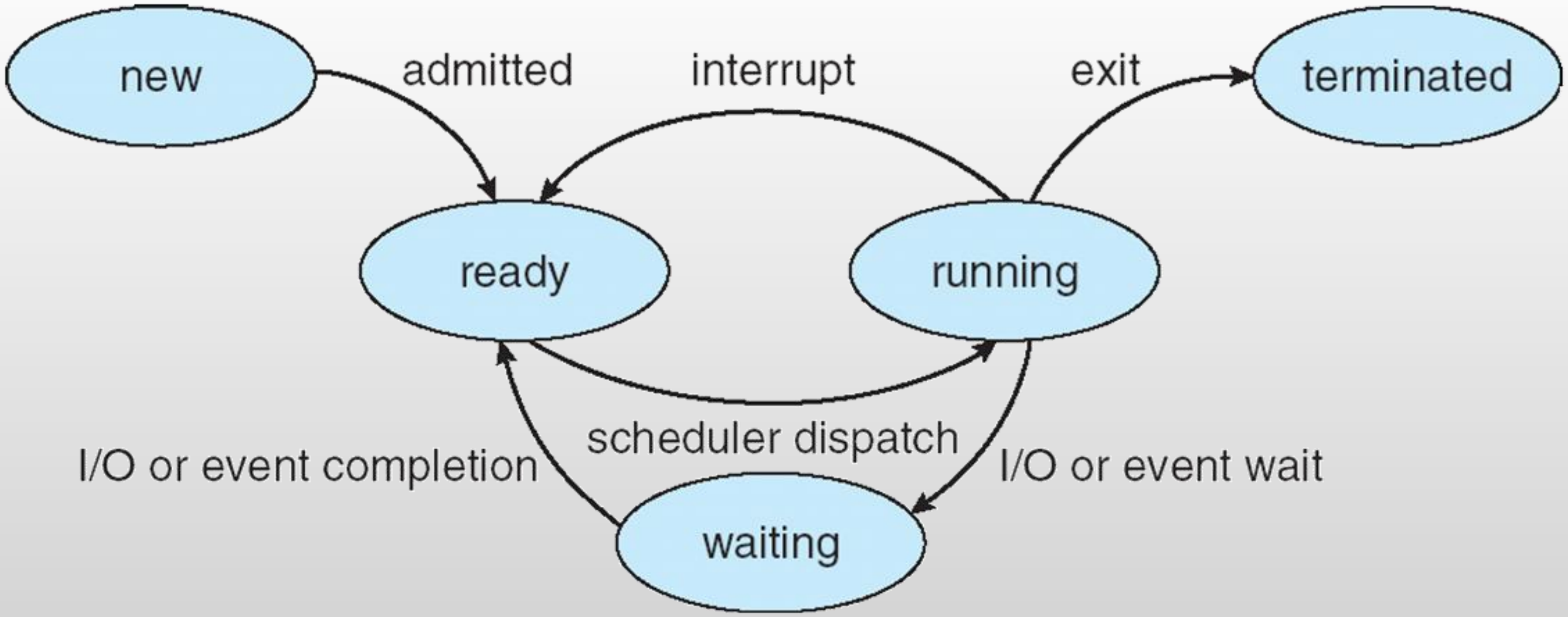
- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - **Text section –** contains the program code
  - **Program counter** – holds the current activity (processor registers included)
  - **Stack -** contains temporary data, such as function parameters, return addresses, local variables
  - **Data section –** contains global variables
  - **Heap -** memory allocated dynamically during process run time
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes: Consider multiple users executing the same program

The Structure of a Process in Memory



Process State

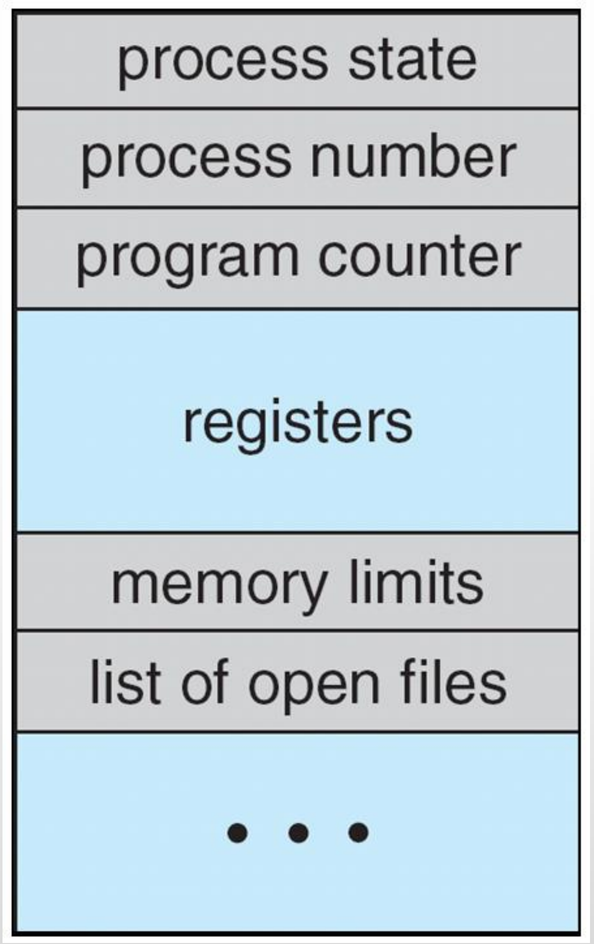
- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur, such as an I/O completion or reception of a signal
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution



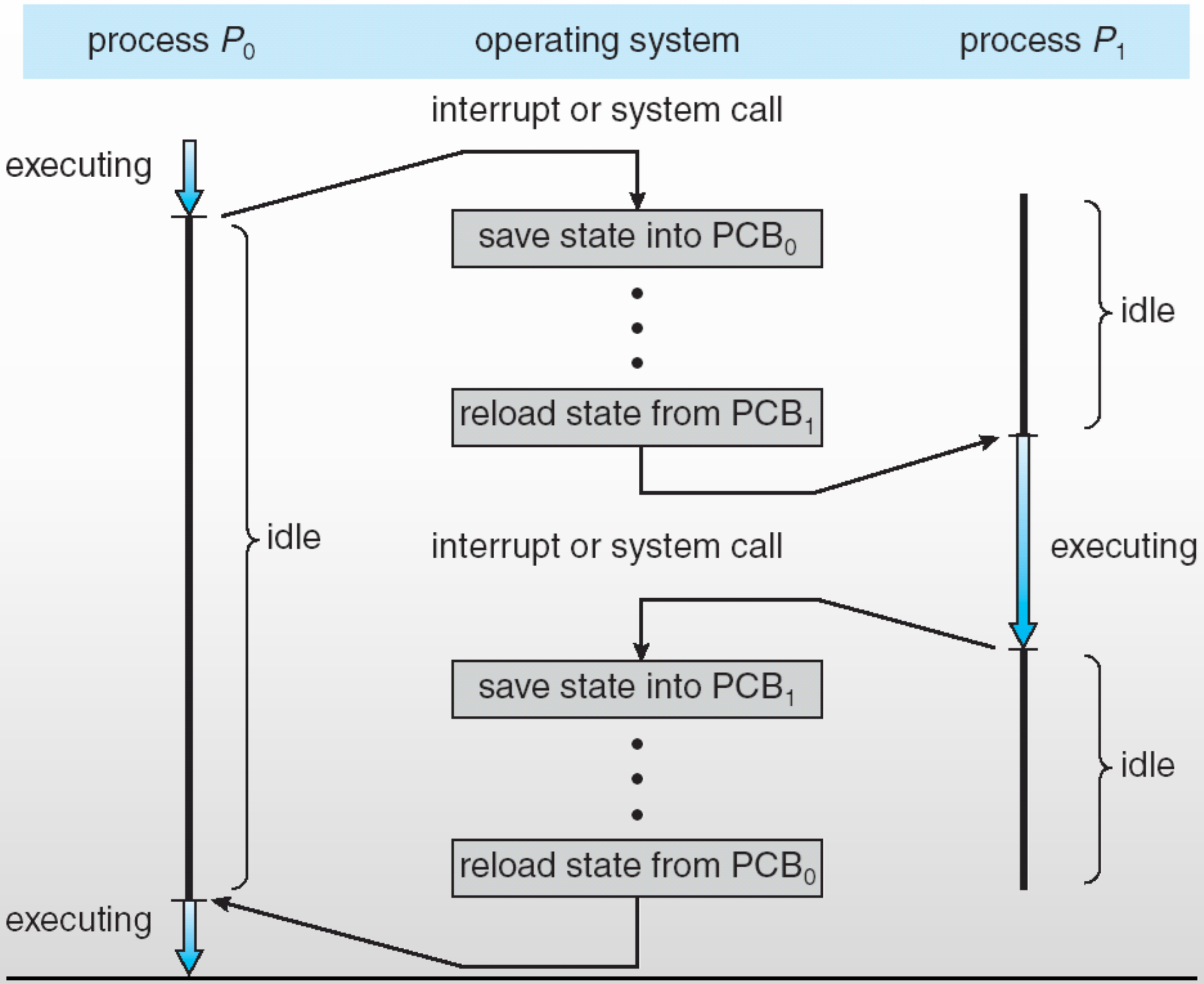
Process Control Block (PCB)

Also called **task control block.** Information associated with each process:

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



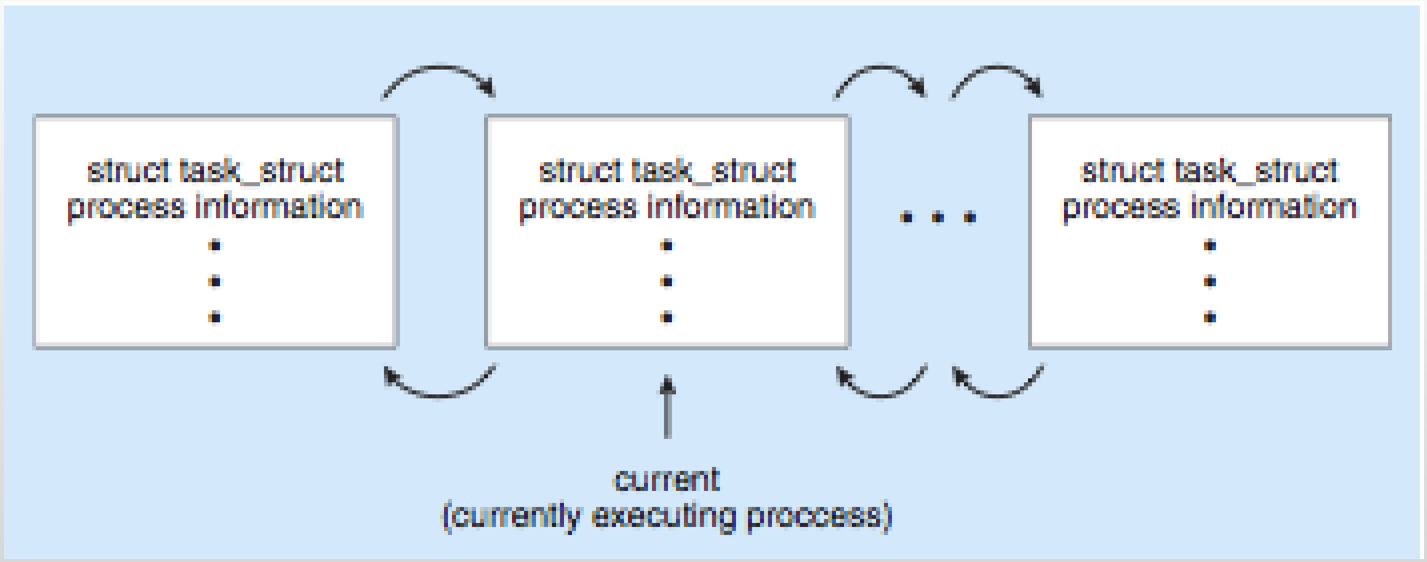
CPU Switch From Process to Process



Process Representation in Linux

Represented by the C structure **`struct task_struct`**

```
pid t_pid;                              /* process identifier */
long state;                             /* state of the process */
unsigned int time_slice                 /* scheduling information */
struct task_struct *parent;             /* this process's parent */
struct list_head children;              /* this process's children */
struct files_struct *files;             /* list of open files */
struct mm_struct *mm;                 /* address space of this process */
```

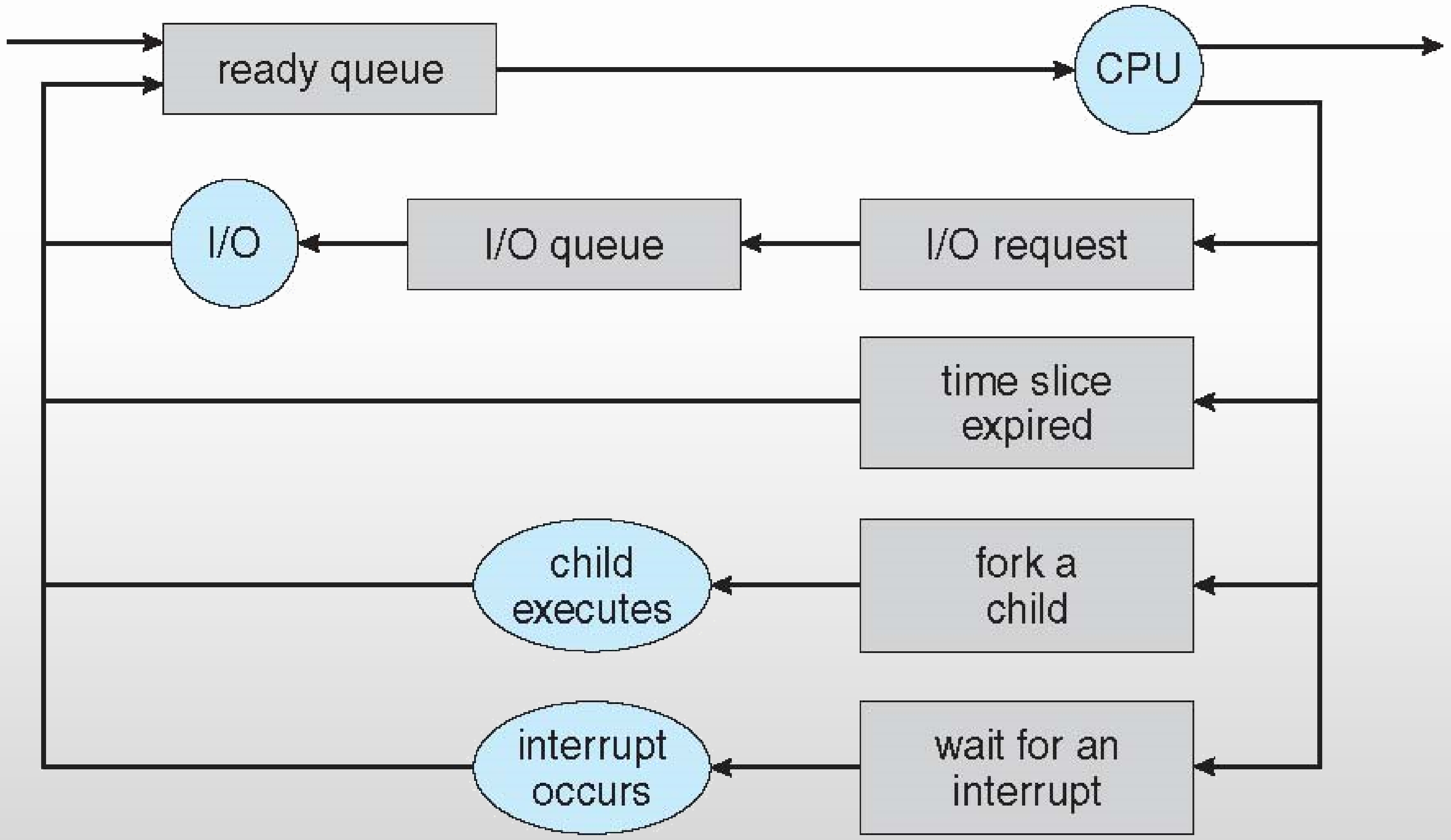


Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows

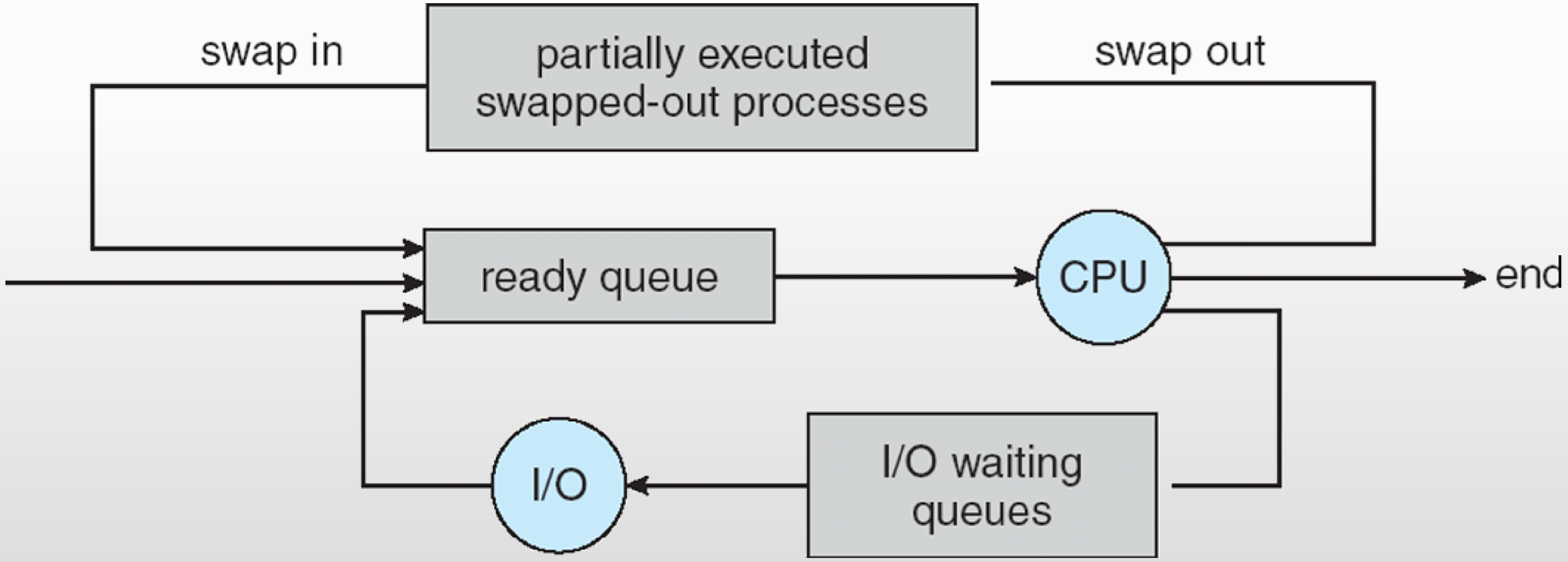


Schedulers

- A process migrates among the various scheduling queues throughout its lifetime.
- **Scheduler** selects processes from different queues
  - **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into memory (ready queue) for execution
  - **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
    - Sometimes the only scheduler in a system
- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (sec, min) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Addition of Medium-Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



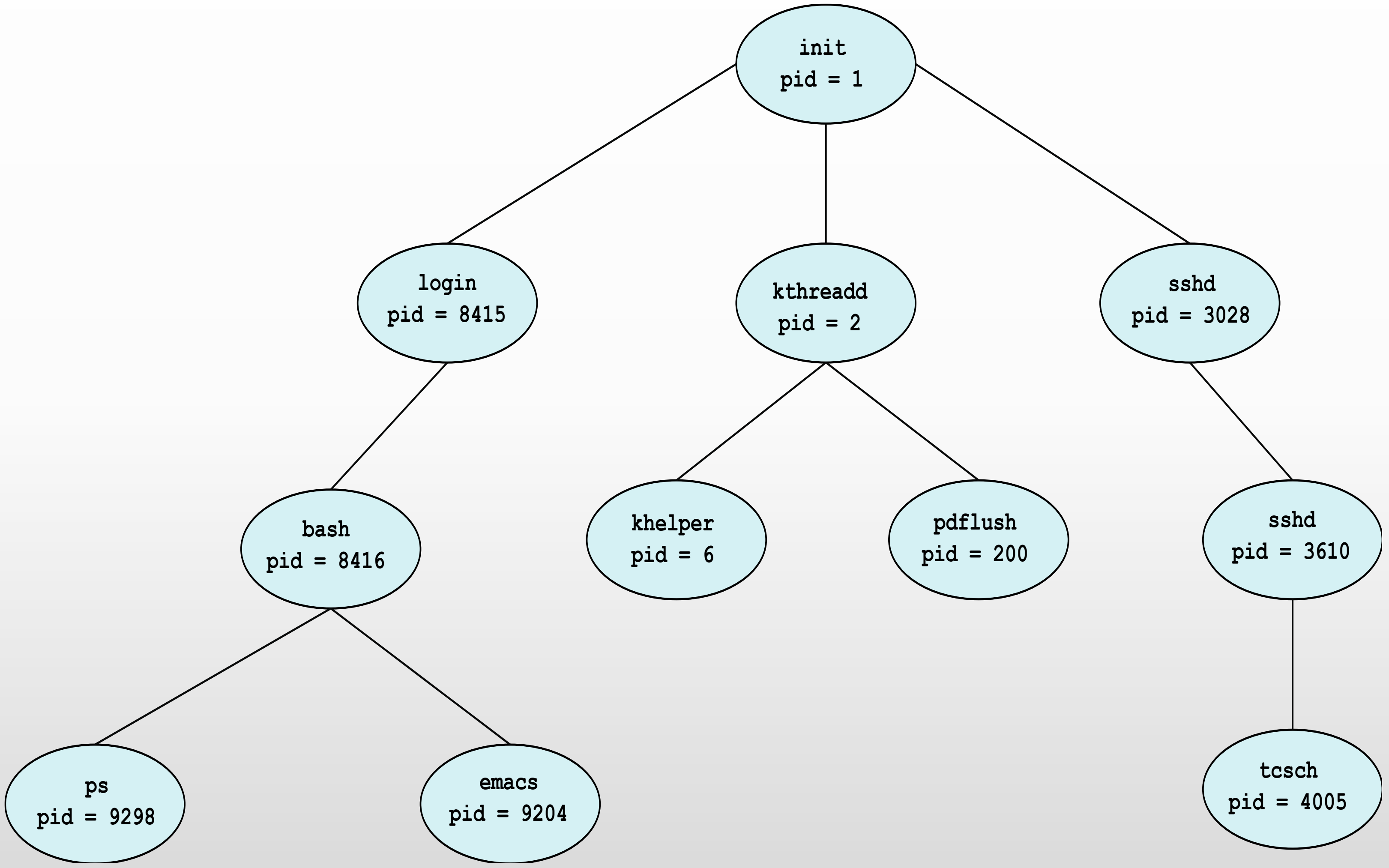
Context Switch

- 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
  - The more complex the OS and the PCB -> longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Process Creation

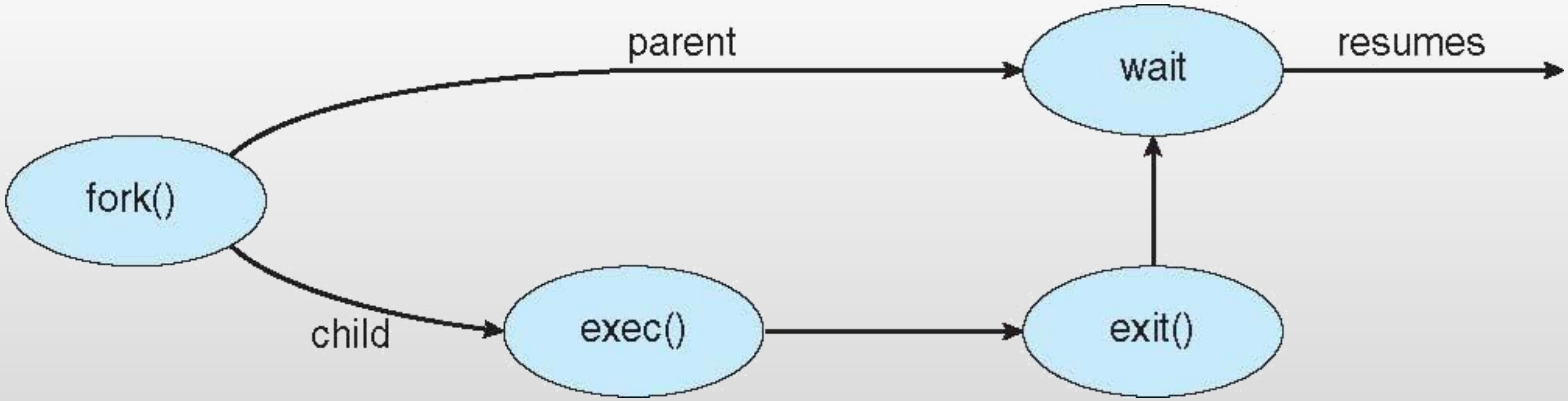
- A process may create several new processes, via a create-process system call, during execution.
- The creating process is called **Parent** process.
- The new processes are called the **children** process.
- Each process in turn create other processes, forming a **tree** of processes.
- Generally, process identified and managed via a **process identifier** (**pid**)
- When a process creates a sub process, it also needs resources.
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

A Tree of Processes in Linux

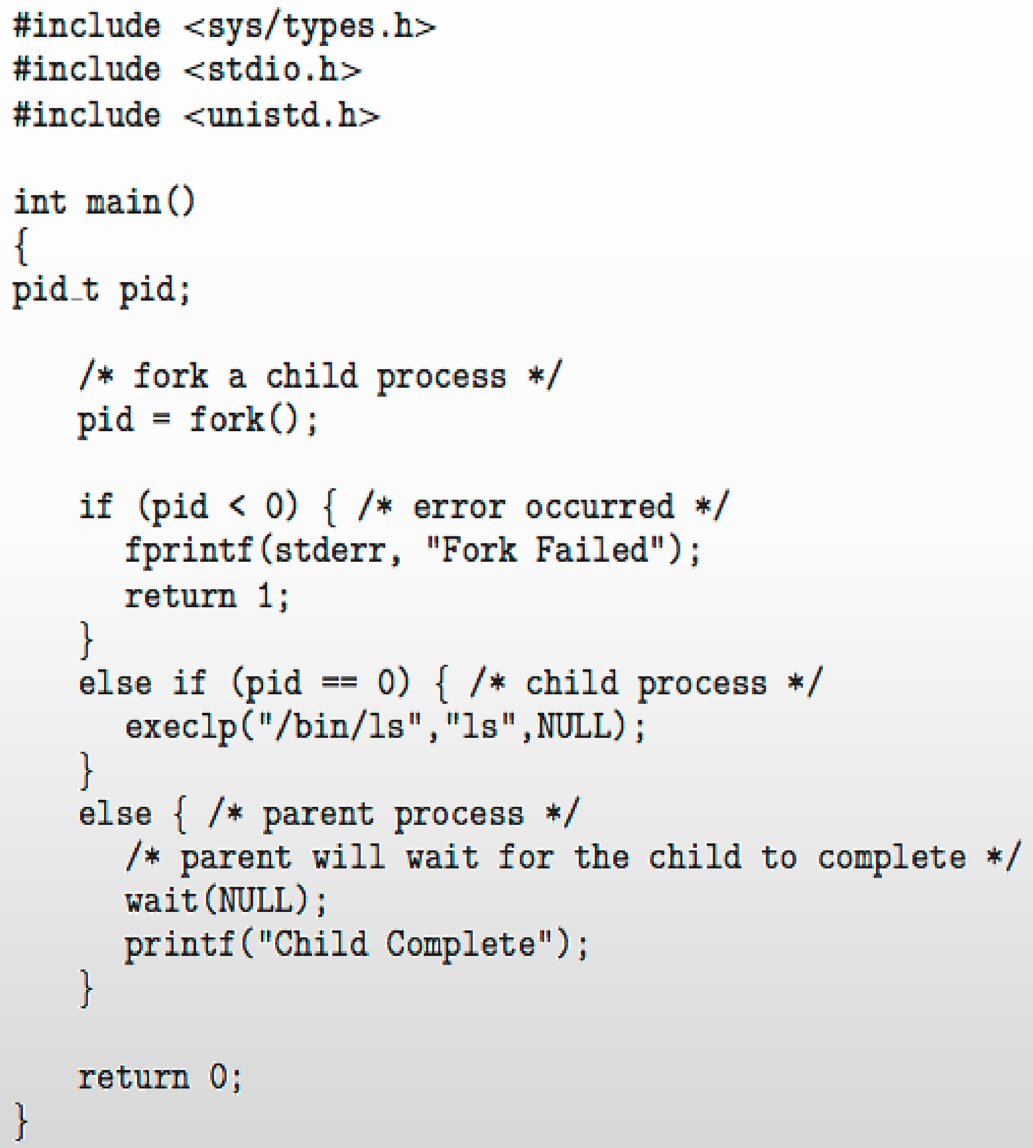


Process Creation (Cont.)

- The address space of the new process may
  - Duplicate of the parent process
  - The child process has a new 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



C Program Forking Separate Process



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

Process Termination

- A process terminate, when it finishes executing its final statement
- It asks the OS for deleting itself by **exit()** system call
- At this point, the process may return the status value to the parent via **wait()** system call
- All the resources used by the **child process** will be deallocated
- Some process may terminate child process by **TerminateProcess()** system call

Process Termination (Cont.)

- A process (**Parent**) may terminate execution of children processes (`abort()`) for variety of reason
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating systems do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

- Wait for termination, returning the pid:

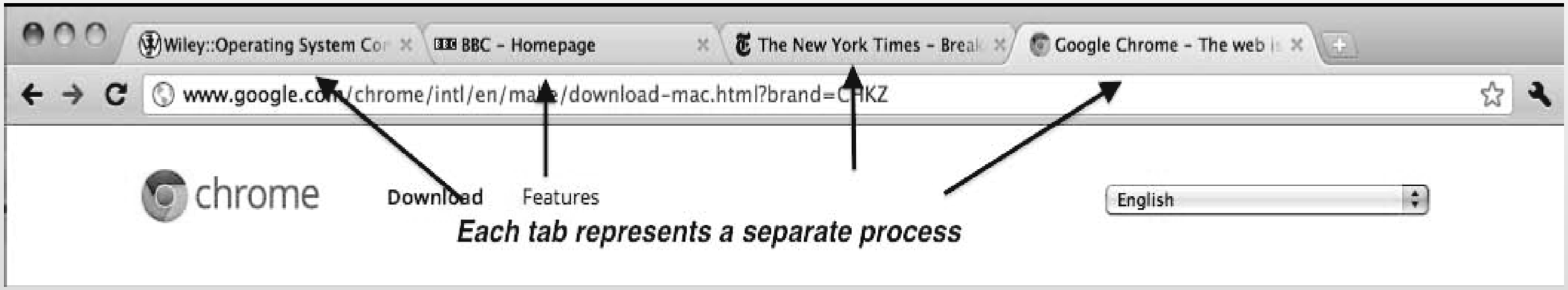
```
pid t_pid; int status;
pid = wait(&status);
```

- If no parent waiting, then terminated process is a **zombie**
- If parent terminated, processes are **orphans**

Multi-process Architecture – Chrome Browser

- Many web browsers ran as single process (some still do)
  - If one web site causes trouble, entire browser can hang or crash

- Google Chrome Browser is multiprocess with 3 categories
  - **Browser** process manages user interface, disk and network I/O
  - **Renderer** process renders web pages, deals with HTML, Javascript, new one for each website opened
  - **Plug-in** process for each type of plug-in

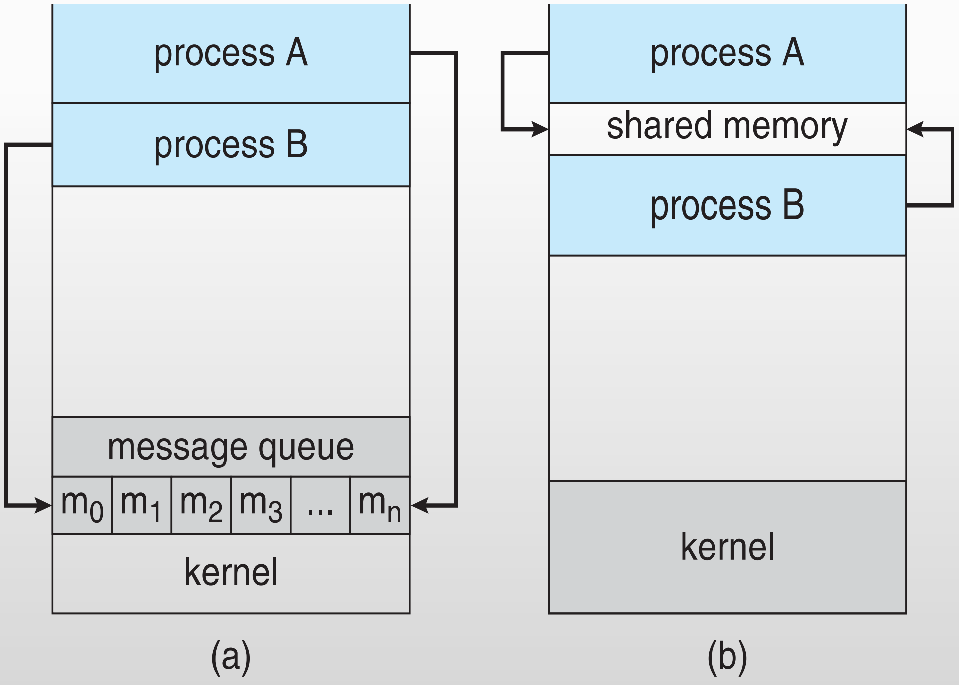

Wiley::Operating System Co
BBC – Homepage
The New York Times – Break
Google Chrome – The web
www.google.com/chrome/intl/en/make/download-mac.html?brand=CHKZ
chrome
Download
Features
English
Each tab represents a separate process

Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- **Cooperating** process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
    - Several users may be interested in the same piece of information
    - Provide an environment to allow concurrent access to such information
  - Computation speedup
    - A particular task can be break into pieces to execute in parallel to run fast
  - Modularity
    - To construct a system in a modular fashion
    - Dividing the system functions into separate processes or threads
  - Convenience
    - User may work on many tasks at the same time
    - For instance, editing, printing, compiling a file

Interprocess Communication

- Cooperating processes need **Inter-process communication** (**IPC**)
- Two models of IPC
  - **Message passing**
    - Communication takes place by means of messages exchanged between the cooperating processes
    - Useful for exchanging smaller amounts of data
  - **Shared memory**
    - A region of memory that is shared by cooperating processes is established
    - Process can then exchange information by reading and writing data to shared region

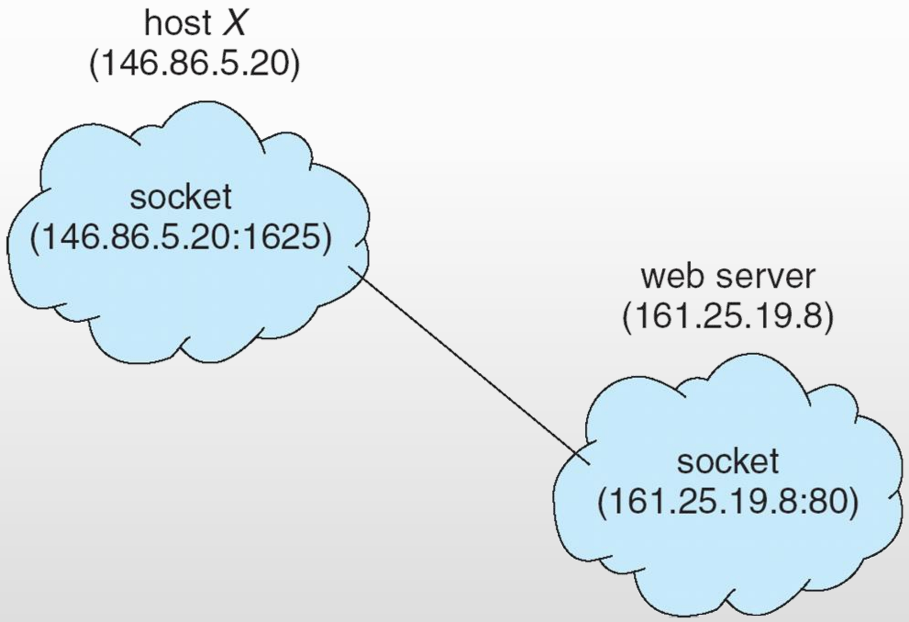


Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)

Sockets

- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are ***well known***, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running



Sockets in Java

- Three types of sockets
  - **Connection-oriented** (**TCP**)
  - **Connectionless** (**UDP**)
  - `MulticastSocket` class– data can be sent to multiple recipients
- Consider this "Date" server:

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

LEADING UNIVERSITY

Question & Discussion

Task Assign

estd. 2001
LEADING UNIVERSITY

Thank You

kohinoor_cse@lus.ac.bd