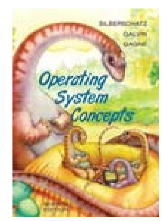
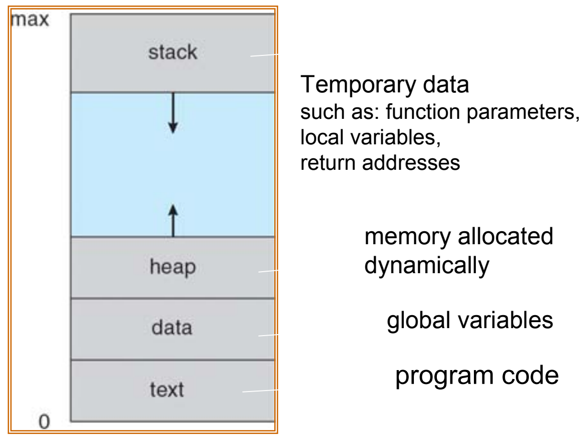
Chapter 3: Processes

- Process Concept (3.1)
- Process Scheduling (3.2)
- Operations on Processes (3.3)
- Interprocess Communication (3.4)
- Communication in Client-Server Systems (3.6)



Process Concept (3.1)

- 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:
  - program counter
  - stack
  - data section



Process in Memory

Process Concept (3.1) (cont.)

- 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.
    - ready: The process is waiting to be assigned to a processor.
    - terminated: The process has finished execution.

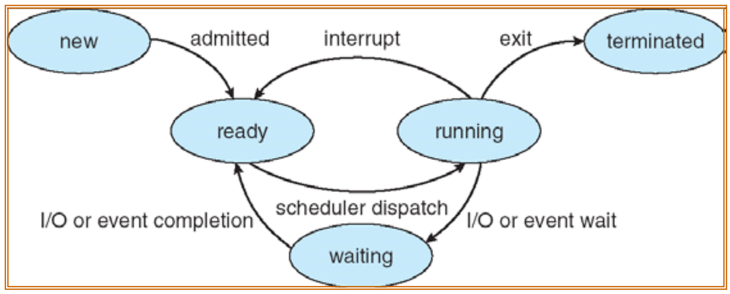
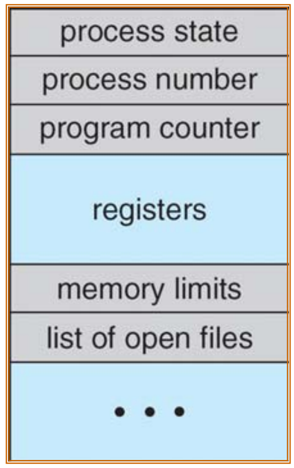


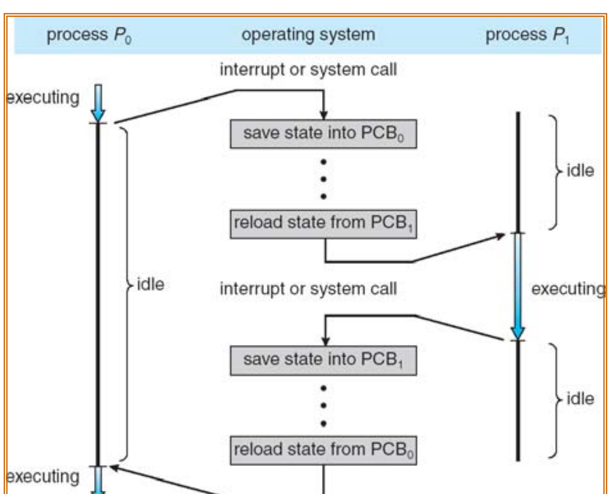
Diagram of Process State

Process Concept (3.1) (cont.)

- Process Control Block (PCB): Information associated with each process.
  - Process state
  - Program counter
  - CPU registers (index registers, stack pointers,…)
  - CPU scheduling information (process priority…
  - Memory-management information (base and limit registers)
  - Accounting information (amount of CPU time used, proc.#)
  - I/O status information (I/O devices for this process)



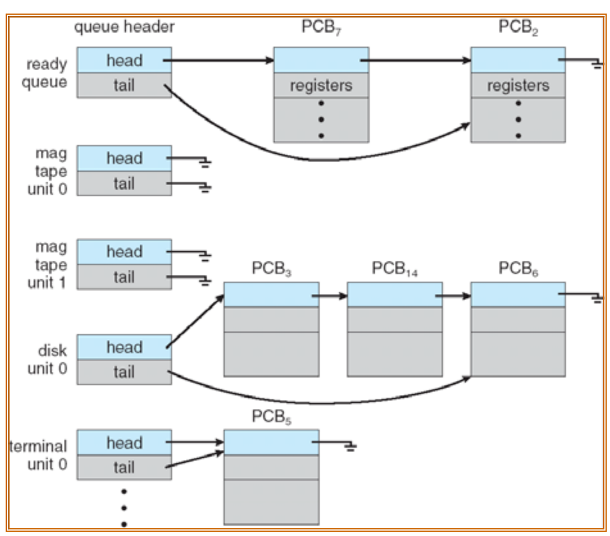
Process Control Block (PCB)



CPU Switch From Process to Process

Process Scheduling (3.2)

- Scheduling queues
  - 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 between the various queues.



Representation of Process Scheduling

Process Scheduling (3.2) (cont.)

- Schedulers
  - Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue from a mass storage.
  - Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU to one of them.

Process Scheduling (3.2) (cont.)

- Schedulers (Cont.)
  - Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast).
  - Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (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.

Process Scheduling (3.2) (cont.)

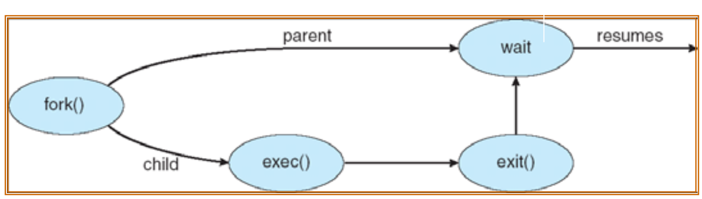
- Context Switch
  - When CPU switches to another process, the operating system must save the state of the old process and load the saved state for the new process.
  - Context-switch time is overhead; the system does no useful work while switching (justify use of threads)
  - Context switch times are dependent on hardware support (memory speed, number of registers, etc.).

Operation on Processes (3.3)

- Process creation
  - Parent process create children processes, which, in turn create other processes, forming a tree of processes.
  - 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.

Operation on Processes (3.3) (cont.)

- Process Creation (Cont.)
  - Address space
    - Child process is a duplicate of the parent process (copy of the address space.)
    - This copy allows the parent process to communicate easily with the child process
    - Child has a program loaded into it.
  - UNIX examples
    - fork system call creates new process (code: 0-child)
    - exec system call (executed by either the parent or the child) used after a fork to replace the process' memory space with a new program.
    - The exec() system call loads a binary file into memory by destroying the memory image of the program containing the exec system call and then starts the execution of the binary file.

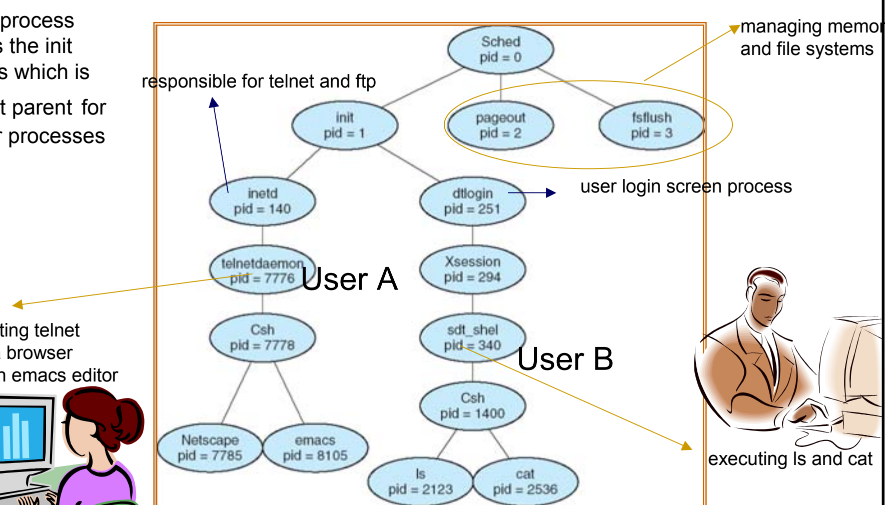


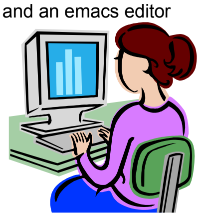
Process Creation

C Program Forking Separate Process

```
int main()
{
pid_t  pid;
   /* fork another process */
   pid = fork();
   if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
   }
   else if (pid == 0) { /* new child process */
        execlp("/bin/ls", "ls", NULL); /* child overlays its
   address space with the command /bin/ls */
   }
   else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);/* return the process identifier of a
   terminated child so that the parent can tell which of its
   possibly many children has terminated */
        printf ("Child Complete");
        exit(0); /* parent terminates */
   }
}
```

Sched process creates the init process which is the root parent for all user processes





A tree of processes on a typical Solaris

Operation on Processes (3.3) (cont.)

- Process Termination
  - Process executes last statement and asks the operating system to decide it (exit).
    - Return of 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.
    - Parent is exiting.
      - Operating system does not allow child to continue if its parent terminates.
      - Cascading termination (if no parent then no children).

Interprocess Communication (3.4)

- Independent process cannot affect or be affected by the execution of another process.
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing (concurrent access to info)
  - Computation speed-up
  - Modularity (separate processes or threads)
  - Convenience (editing, printing at the same time)

Interprocess Communication (3.4) (cont.)

- There are two models of interprocess communication: (1) shared memory and (2) message passing
- (1) Shared memory:
  - Producer-Consumer Problem
  - Paradigm for cooperating processes,
  - producer process produces information that is consumed by a consumer process.
    - unbounded-buffer places no practical limit on the size of the buffer.
    - bounded-buffer assumes that there is a fixed buffer size.

Interprocess Communication (3.4) (cont.)

- Bounded-Buffer – Shared-Memory Solution
  - Shared data= buffer (created through system calls)

```
#define BUFFER_SIZE 10 /* circular array*/
          Typedef struct {
            . . .
          } item;
          item buffer[BUFFER_SIZE];
          int in = 0;
          int out = 0;
```

Interprocess Communication (3.4) (cont.)

- Bounded-Buffer – Inset() Method (Producer process)

```
item nextProduced;
while (true) {
   /* Produce an item in nextProduced */
while (((in = (in + 1) % BUFFER SIZE count) == out)
      ;   /* do nothing -- no free buffers */
     buffer[in] = nextProduced;
     in = (in + 1) % BUFFER SIZE;
    {
```

Interprocess Communication (3.4) (cont.)

- Bounded-Buffer – Remove () Method (Consumer process)

```
item nextConsumed;
while (true) {
        while (in == out)
              ; // do nothing -- nothing to consume

     nextConsumed = buffer[out];
     out = (out + 1) % BUFFER SIZE;
/* consume the item in nextConsumed */
   {
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements at the same time
- This shared buffer should be implemented by the application programmer.

Interprocess Communication (3.4) (cont.)

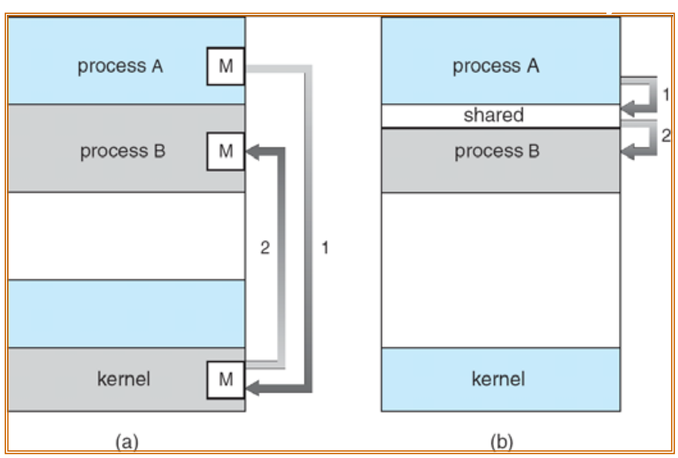
- OS mechanism for processes to communicate and to synchronize their actions without sharing the same address space.
- (2) Message passing – processes communicate with each other without resorting to shared variables such as "buffer".
- IPC facility provides two operations:
  - send(message) – message size fixed or variable
  - receive(message)

Interprocess Communication (3.4) (cont.)

- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties: our concern !)

Interprocess Communication (3.4) (cont.)

- Implementation Questions
  - How are links established?
  - Can a link be associated with more than two processes?
  - How many links can there be between every pair of communicating processes?
  - What is the capacity of a link?
  - Is the size of a message that the link can accommodate fixed or variable?
  - Is a link unidirectional or bi-directional?



Communications Models

Interprocess Communication (3.4) (cont.)

- Direct Communication
  - Processes must name each other explicitly:
    - send (P, message) – send a message to process P
    - receive(Q, message) – receive a message from process Q
  - Properties of communication link
    - Links are established automatically (need process id's).
    - A link is associated with exactly one pair of communicating processes.
    - Between each pair there exists exactly one link.
    - The link may be unidirectional, but is usually bi-directional.

Interprocess Communication (3.4) (cont.)

- Indirect Communication
  - Messages are directed and received from mailboxes (also referred to as ports).
    - Each mailbox has a unique id.
    - Processes can communicate only if they share a mailbox.
  - Properties of communication link
    - Link established only if processes share a common mailbox
    - Each pair of processes may share several communication links.
    - Link may be unidirectional or bi-directional.

Interprocess Communication (3.4) (cont.)

- Indirect Communication (cont.)
  - Operations
    - create a new mailbox
    - send and receive messages through mailbox
    - destroy a mailbox
  - Primitives are defined as:
  - send(A, message) – send a message to mailbox A
  - receive(A, message) – receive a message from mailbox A

Interprocess Communication (3.4) (cont.)

- Indirect Communication (cont.)
  - Mailbox sharing
    - P1 , P2 , and P3 share mailbox A.
    - P1 , sends; P2 and P3 receive.
    - Who gets the message?
  - Solutions
    - Allow a link to be associated with at most two processes.
    - Allow only one process at a time to execute a receive operation.
    - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Interprocess Communication (3.4) (cont.)

- Synchronization
  - Message passing may be either blocking or non-blocking
  - Blocking is considered synchronous
    - Blocking send has the sender block until the message is received
    - Blocking receive has the receiver block until a message is available
  - Non-blocking is considered asynchronous
    - Non-blocking send has the sender send the message and continue
    - Non-blocking receive has the receiver receive a valid message or null

Interprocess Communication (3.4) (cont.)

- Buffering
  - Queue of messages attached to the link; implemented in one of three ways.

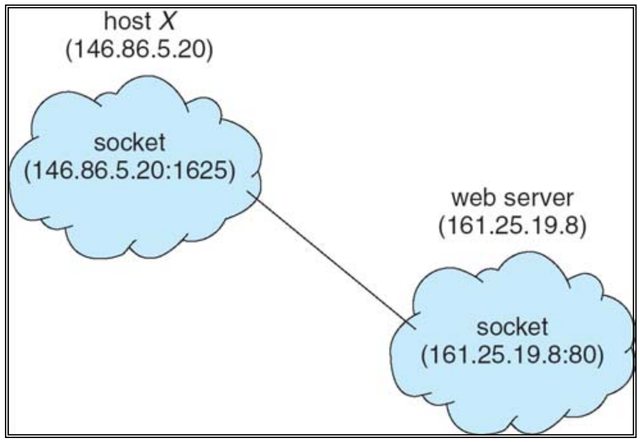
    1. Zero capacity – 0 messages
       Sender must wait for receiver (rendezvous).
    2. Bounded capacity – finite length of n messages
       Sender must wait if link full.
    3. Unbounded capacity – infinite length
       Sender never waits.

Communication in Client-Server Systems (3.6)

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

Communication in Client-Server Systems (3.6) (cont.)

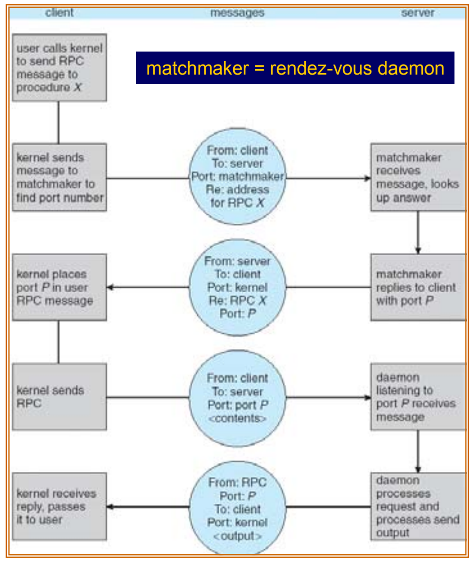
- Sockets
  - A socket is defined as an endpoint for communication.
  - Concatenation of IP address and port
  - The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
  - Communication consists between a pair of sockets.



Socket Communication

Communication in Client-Server Systems (3.6) (cont.)

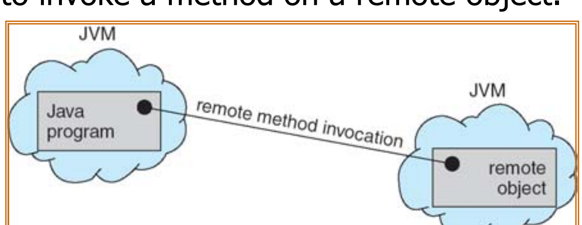
- Remote Procedure Calls (RPC)
  - Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
  - Stubs – client-side proxy TO USE the actual procedure on the server.
  - The client-side stub locates the server and *marshalls* the parameters (packaging the parameters into a form that can be transmitted over a network).
  - The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server.

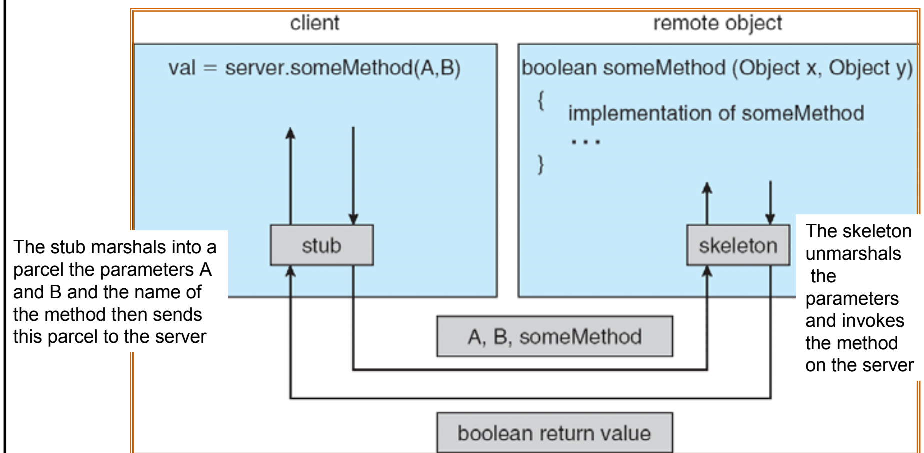


Execution of RPC

Communication in Client-Server Systems (3.6) (cont.)

- Remote Method Invocation
  - Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
  - RMI allows a Java program on one machine to invoke a method on a remote object.





Marshalling Parameters