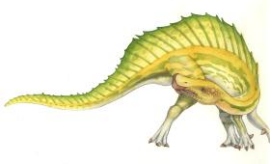
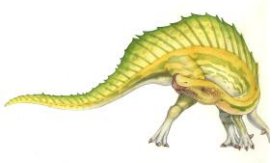
Interprocess Communication – Message Passing

- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive



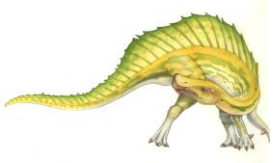
Implementation Questions

- 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?



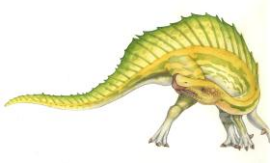
Direct Communication

- 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
  - 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



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
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

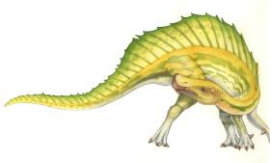


Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

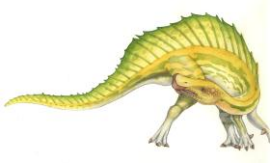
  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A



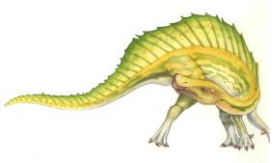
Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 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.



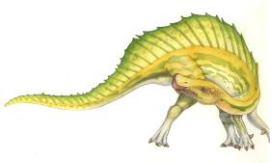
Synchronization

- 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



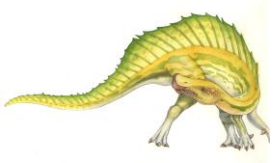
Buffering

- 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



Message Passing in Linux

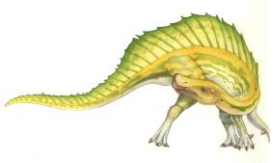
- Pipes permit sequential communication from one process to a related process.
- FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the filesystem.
- Sockets support communication between unrelated processes even on different computers



Pipes

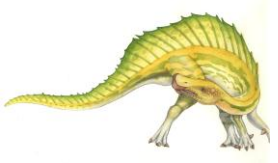
- To create a pipe, invoke the pipe command. Supply an integer array of size 2. The call to pipe stores the reading file descriptor in array position 0 and the writing file descriptor in position 1. Data written to the file descriptor read_fd can be read back from write_fd.

```
int pipe_fds[2];
int read_fd;
int write_fd;
pipe (pipe_fds);
read_fd = pipe_fds[0];
write_fd = pipe_fds[1];
```



Pipes

- A call to pipe creates file descriptors, which are valid only within that process and its children. Thus, pipes can connect only related processes.
- When you invoke the command ls | less, two forks occur: one for the ls child process and one for the less child process. Both of these processes inherit the pipe file descriptors so they can communicate using a pipe



FIFOs

- A first-in, first-out (FIFO) file is a pipe that has a name in the files ystem. Any process can open or close the FIFO; the processes on either end of the pipe need not be related to each other. FIFOs are also called named pipes.

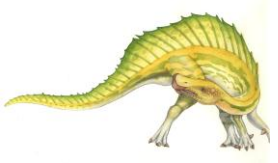
  mkfifo temp

  ls –l temp

  In one teminal, type cat < temp

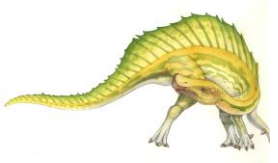
  In another terminal, type cat > temp, then type something

  rm temp

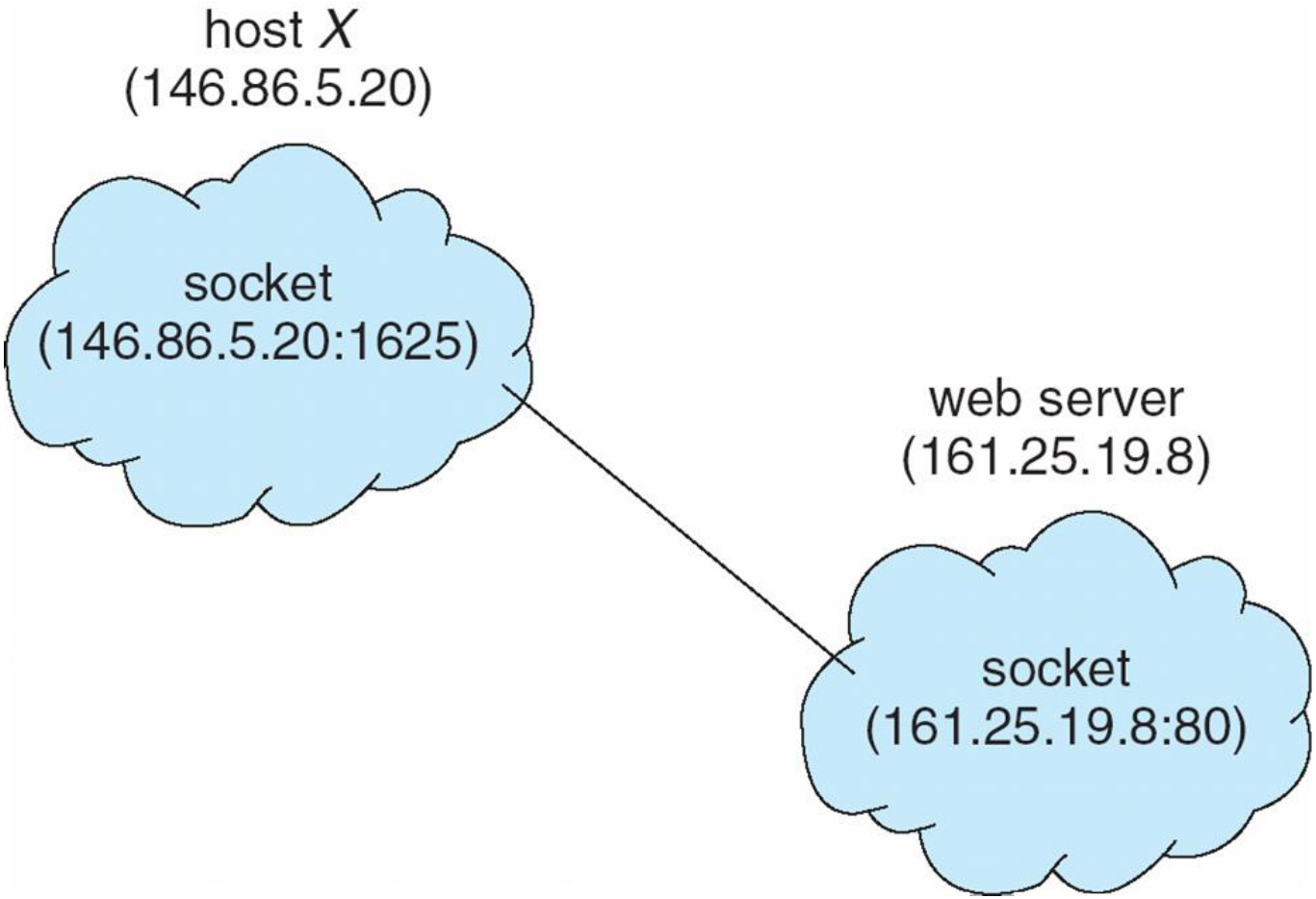


Sockets

- A socket is defined as an *endpoint for communication*
  - A socket is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machines.
- 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



Sockets

- Internet programs such as Telnet, rlogin, FTP, talk, and the World Wide Web use sockets.

telnet www.wmich.edu 80

Trying xxx.xx.xx.x...

Connected to …

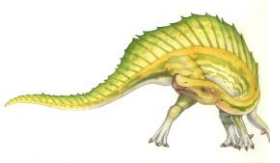
Escape character is '^]'.

GET /

<html>

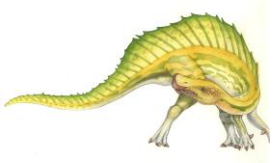
<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">



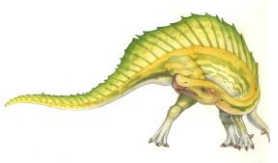
Socket Types

- Connection styles guarantee delivery of all packets in the order they were sent. If packets are lost or reordered by problems in the network, the receiver automatically requests their retransmission from the sender.
- Datagram styles do not guarantee delivery or arrival order. Packets may be lost or reordered in transit due to network errors or other conditions. Each packet must be labeled with its destination and is not guaranteed to be delivered.The system guarantees only “best effort,” so packets may disappear or arrive in a different order than shipping.



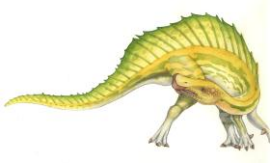
Creating Sockets

- When you create a socket, specify the three socket choices: namespace, communication style, and protocol.
  - For the namespace, use constants beginning with PF_ (abbreviating “protocol families”). E.g , PF_LOCAL or PF_UNIX specifies the local namespace, and PF_INET specifies Internet namespaces.
  - For the communication style, use constants beginning with SOCK_. Use SOCK_STREAM for a connection-style socket, or use SOCK_DGRAM for a datagram-style socket.
  - For the protocol, specifies the low-level mechanism to transmit and receive data. Each protocol is valid for a particular namespace-style combination. Because there is usually one best protocol for each such pair, specifying 0 is usually the correct protocol.
  - If socket succeeds, it returns a file descriptor for the socket. You can read from or write to the socket using read, write, and so on, as with other file descriptors.
  - When you are finished with a socket, call close to remove it.



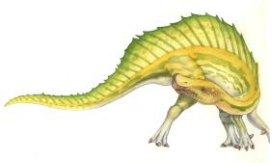
Connecting Sockets

- A client is the process initiating the connection, and a server is the process waiting to accept connections.
- the client calls connect, specifying the address of a server socket to connect to
- The client calls connect to initiate a connection from a local socket to the server socket specified by the second argument.
- The third argument is the length, in bytes, of the address structure pointed to by the second argument.
- Socket address formats differ according to the socket namespace.



Servers

- A server's life cycle consists of the creation of a connection-style socket, binding an address to its socket, placing a call to listen that enables connections to the socket, placing calls to accept incoming connections, and then closing the socket.



End of Chapter 3