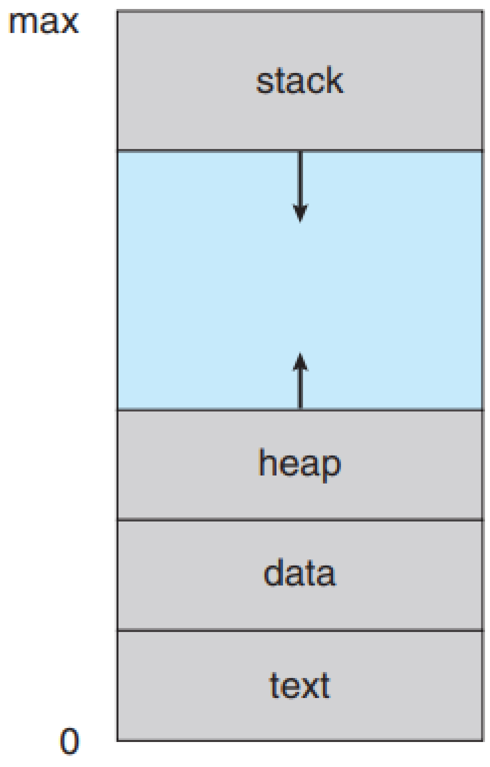
Reading assignment 1: Chapter 3 (Processes)

Processes Concept

The processes

- a program in execution
- the status of the current program is represented by a value named "program counter" and the contents of the processor's registers.
- memory layout of a process



    - Text section: the execution code
    - Data section: global variables
    - Heap section: memory that dynamically allocated during program run time.
    - Stack section: temporary memory storage when invoking functions.

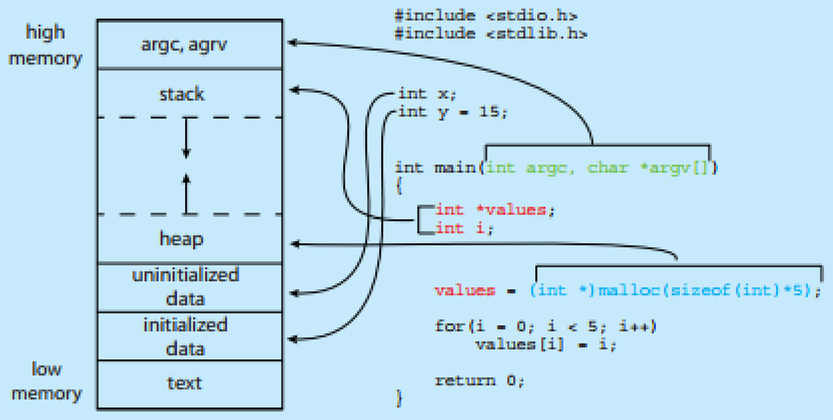
Process States

- as a program executes, it changes state
    - the state of the process is defined in part by the current activity of that process.

MEMORY LAYOUT OF A C PROGRAM

The figure shown below illustrates the layout of a C program in memory, highlighting how the different sections of a process relate to an actual C program. This figure is similar to the general concept of a process in memory as shown in Figure 3.1, with a few differences:

- The global data section is divided into different sections for (a) initialized data and (b) uninitialized data.
- A separate section is provided for the `argc` and `argv` parameters passed to the `main()` function.

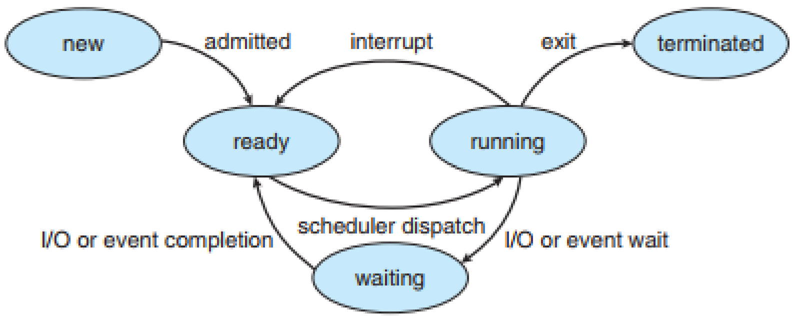


```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

The GNU `size` command can be used to determine the size (in bytes) of some of these sections. Assuming the name of the executable file of the above C program is `memory`, the following is the output generated by entering the command `size memory`:

```
text    data    bss    dec     hex    filename
1158    284     8      1450    5aa    memory
```

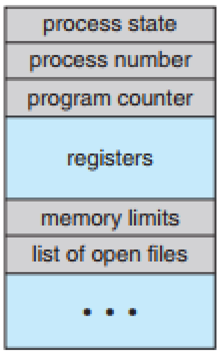
The `data` field refers to uninitialized data, and `bss` refers to initialized data. (`bss` is a historical term referring to ***block started by symbol***.) The `dec` and `hex` values are the sum of the three sections represented in decimal and hexadecimal, respectively.



- 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 process.
- Terminated: The process has finished execution.

Process Control Block

- Each process is represented in the operating system by a process control block (PCB) - also called a task controlled list.



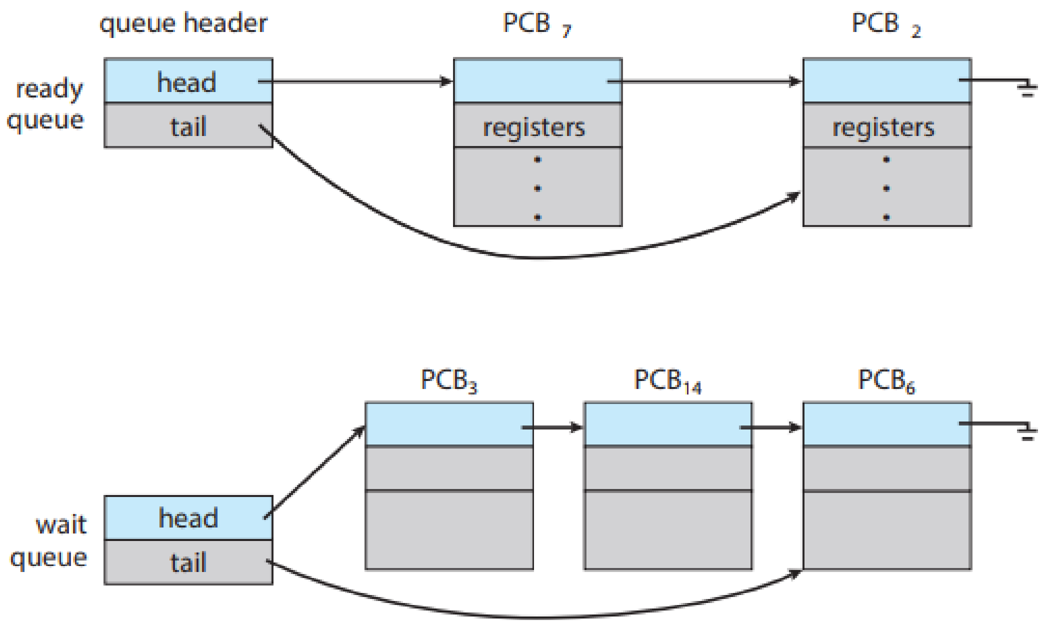
  - Process state: The state may be new, ready, running, waiting, halted, and so on.
  - Program counter: The counter indicates the address of the next instruction to be executed for this process.
  - CPU registers: vary in number and type, it contains accumulators, index registers, stack pointer and general-purpose registers.
  - CPU Scheduling information: include process priority, pointers to scheduling queues and any scheduling para
  - Accounting information: include amount of CPU and real time used, time limits, account numbers, job or process number
  - I/O status information: include list of I/O devices allocated to process, list of file open and so on.

Threads

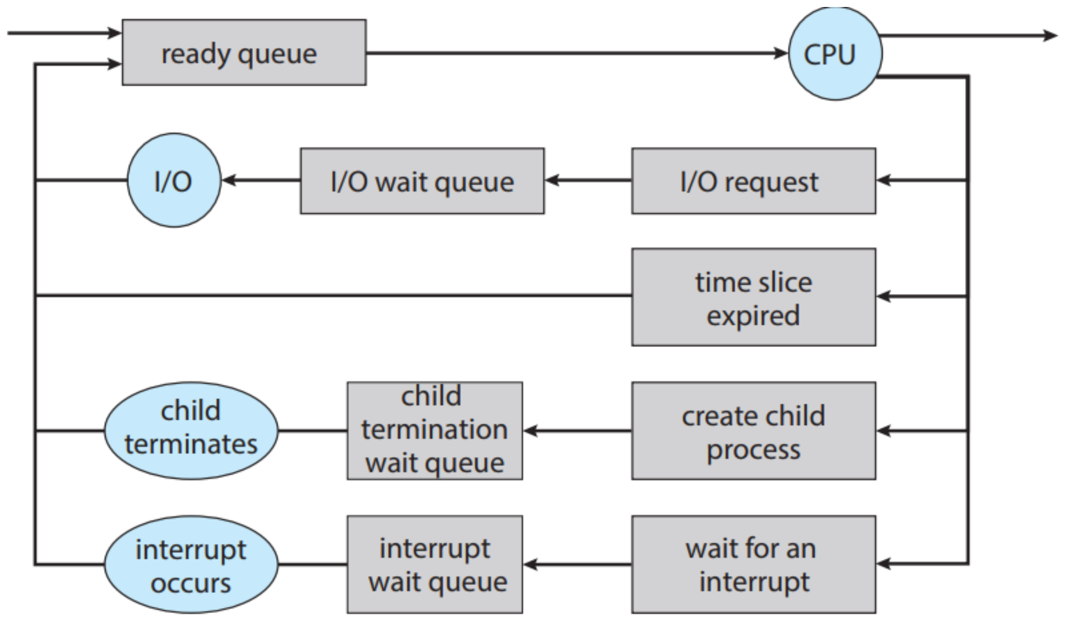
- The process model discussed has implied that ***a process*** is a program that performs a single ***thread*** of execution

Process Scheduling

- The objective of multiprogramming is to have some process running at all times so as to maximize CPU utilization. The objective of time sharing is to switch a CPU core among processes so frequently that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on a core.



Scheduling Queues

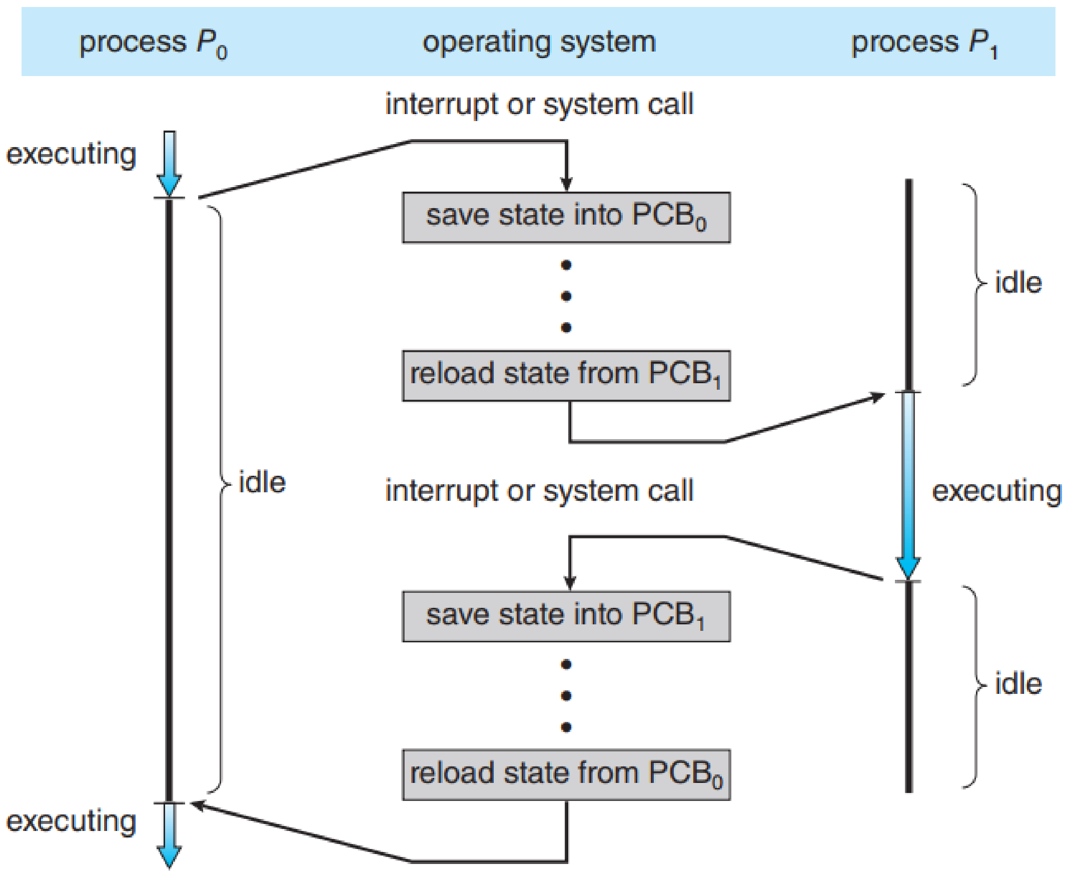
- 

- The process could issue an I/O request and then be placed in a I/O wait queue
- The process could create a new child process and then be placed in a wait queue while it awaits (wait) the child's termination
- The process could be removed forcibly from the core, as a result of an interrupt or having its time slice expire, and be put back in the ready queue

CPU Scheduling

- A process migrates among the ready queue and various wait queues throughout its lifetime
- The role of the CPU scheduler is to select from among the processes that are in the ready queue and allocate a CPU core to one of them. The CPU scheduler must select a new process for the CPU frequently. An I/O-bound process may execute for only a few milliseconds before waiting for an I/O request
- Some operating systems have an intermediate form of scheduling, known as swapping, whose key idea is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming

Context Switch

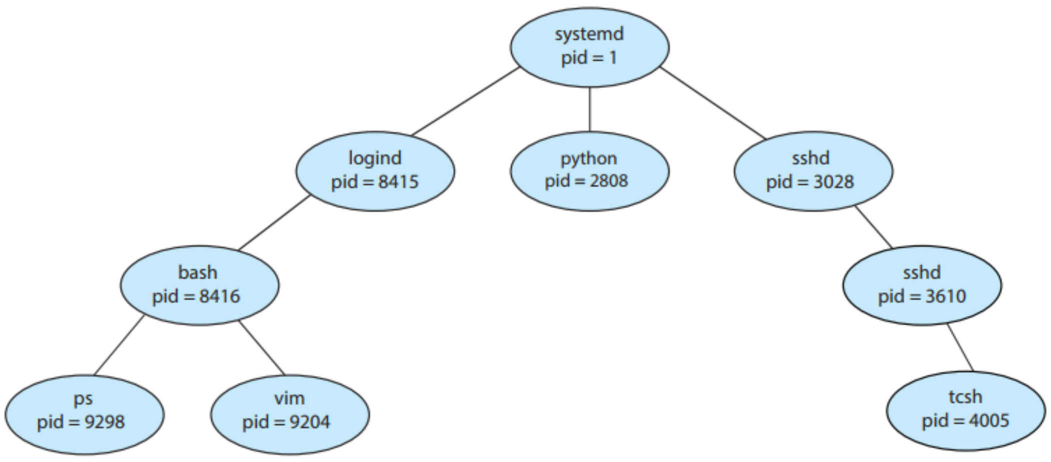
- 

- the switching the CPU core to another process, requires performing a state save of the current process and a state restore of a different process

Operations on Processes

Process Creation

- During the course of execution, a process may create several new processes. As mentioned earlier, the creating process is called a parent process, and the new processes are called the children of that process. Each of these new processes may in turn create other processes, forming a tree of processes.



Process Termination

- The child has exceeded its usage of some of the resources that it has been allocated. (To determine whether this has occurred, the parent must have a mechanism to inspect the state of its children.)
- The task assigned to the child is no longer required
- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates

Interprocess Communication

- Processes executing concurrently in the operating system may be either independent processes or cooperating processes. A process is independent if it does not share data with any other processes executing in the system. A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process
    - Information sharing. Since several applications may be interested in the same piece of information (for instance, copying and pasting), we must provide an environment to allow concurrent access to such information.
    - Computation speedup. If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.
    - Modularity. We may want to construct the system in a modular fashion, dividing the system function.

IPC In Shared-Memory Systems

- Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.
- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space.

IPC In Message-Passing Systems

- If processes *P* and *Q* want to communicate, they must send messages to and receive messages from each other: a ***communication link*** must exist between them

- Here are several methods for logically implementing a link and the send()/receive() operations:

  +) Direct or indirect communication.

  +) Synchronous or asynchronous communication

  +) Automatic or explicit buffering

Naming

- Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication
- With indirect communication, the messages are sent to and received from mailboxes, or ports.
- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed

- Each mailbox has a unique identification.

Synchronization

- Message passing may be either blocking or nonblocking – also known as synchronous and asynchronous.
    - Blocking send: the sending process is blocked until the message is received by the receiving process or by mailbox
    - Nonblocking send: The sending process sends the message and resumes operation
    - Blocking receive: the receiver blocks until a message is available
    - Nonblocking receive: the receiver retrieves either a valid message or a null.

Buffering

- Basically queues can be implemented in 3 ways:
    - Zero capacity: The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message
    - Bounded capacity: The queue has finite length n; thus at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue and the sender can continue execution without waiting
    - Unbounded capacity: The queue's length is potentially infinite, thus any number of messages can wait in it

Examples of IPC Systems

POSIX Shared Memory

- POSIX shared memory is organized using memory-mapped files, which associate the region of shared memory with a file

Watch Message Watching

- Mach was especially designed for distributed systems, but was shown to be suitable for desktop and mobile systems as well, as evidenced by its inclusion in the macOS and iOS operating systems
- The Mach kernel supports the creation and destruction of multiple tasks, which are similar to processes but have multiple threads of control and fewer associated resources
- Most communication in Mach—including all intertask communication—is carried out by messages
- Messages are sent to, and received from, mailboxes, which are called ports in Mach. Ports are finite in size and unidirectional; for two-way communication, a message is sent to one port, and a response is sent to a separate reply port.

Windows

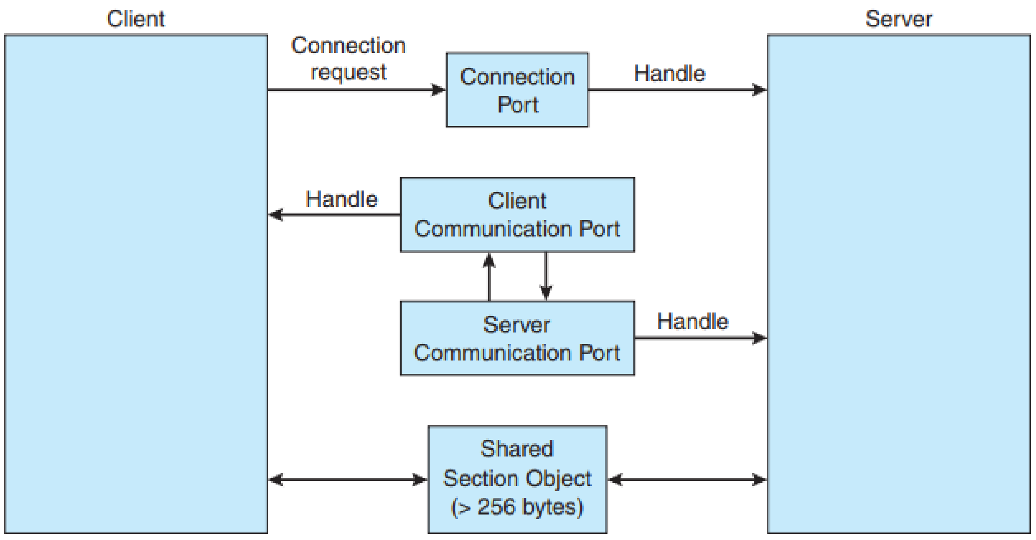
- 
Client
Server
Connection request
Connection Port
Handle
Handle
Client Communication Port
Server Communication Port
Handle
Shared Section Object (> 256 bytes)

Figure 3.19 Advanced local procedure calls in Windows.

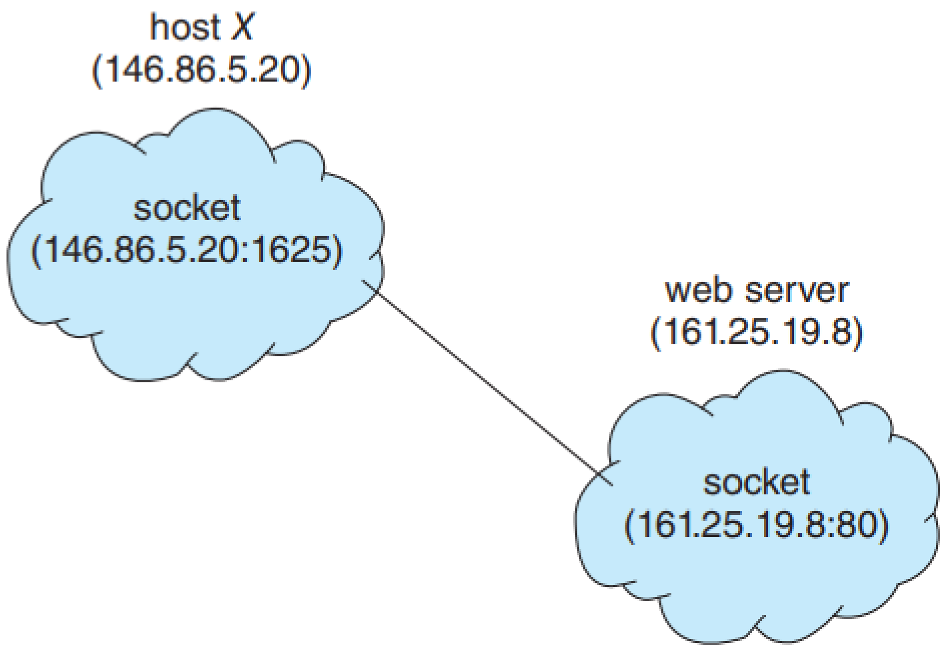
Pipes

- A pipe acts as a conduit allowing two processes to communicate. Pipes were one of the first IPC mechanisms in early UNIX systems

Communication In Client-Server Systems

Socket

- Socket is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets – one of each process
- A socket is identified by an IP address concatenated with a port number. In general, sockets use a client–server architecture. The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection. Servers implementing specific services (such as SSH, FTP, and HTTP) listen to well-known ports (an SSH server listens to port 22; an FTP server listens to port 21; and a web, or HTTP, server listens to port 80). All ports below 1024 are considered well known and are used to implement standard services.



Remote Procedure Calls

- One of the most common forms of remote service is the RPC paradigm, which was designed as a way to abstract the procedure-call mechanism for use between systems with network connections

##