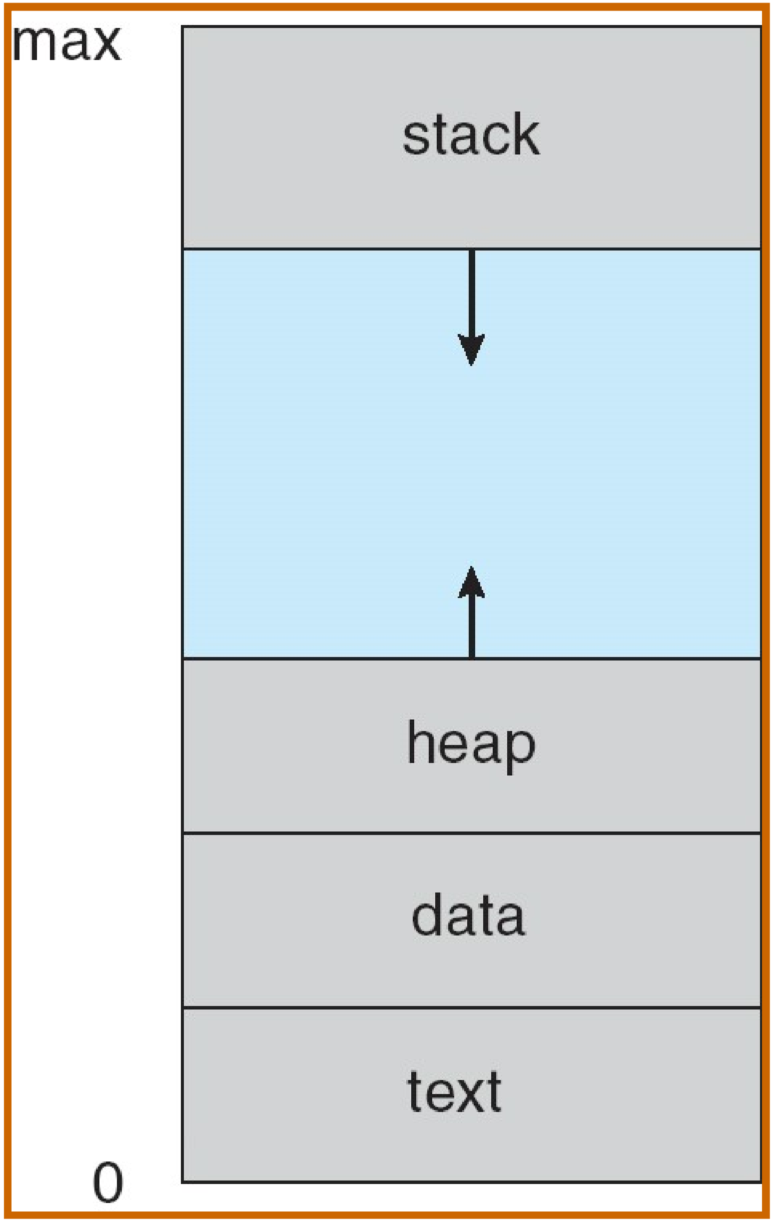
Chapter 3: Process Concept

-

- The Slide does not contain all the information and cannot be treated as a study material for Operating System. Please refer the text book for exams.

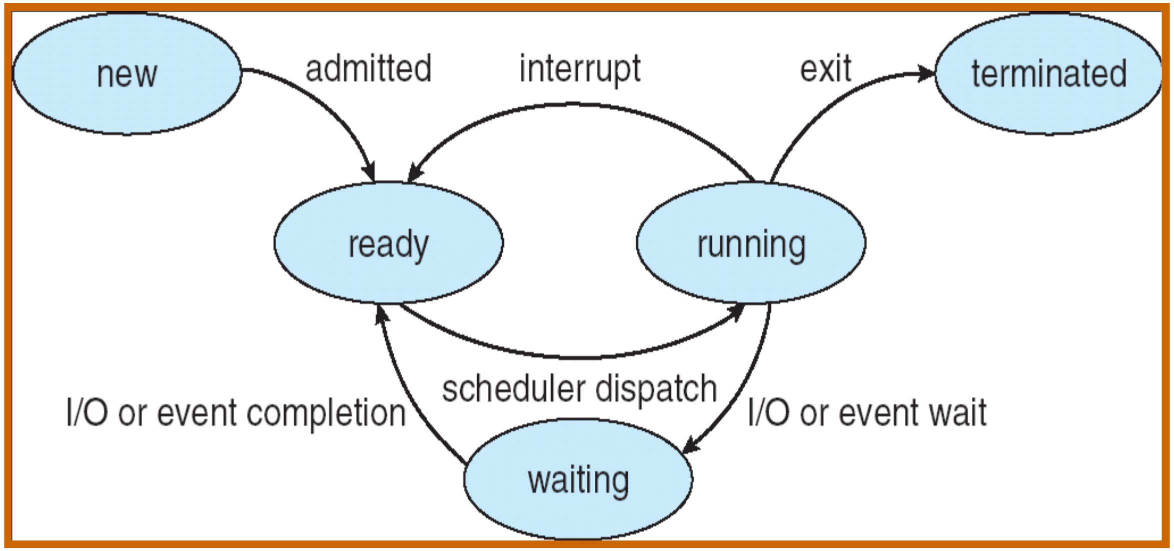
Topics to be Covered

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

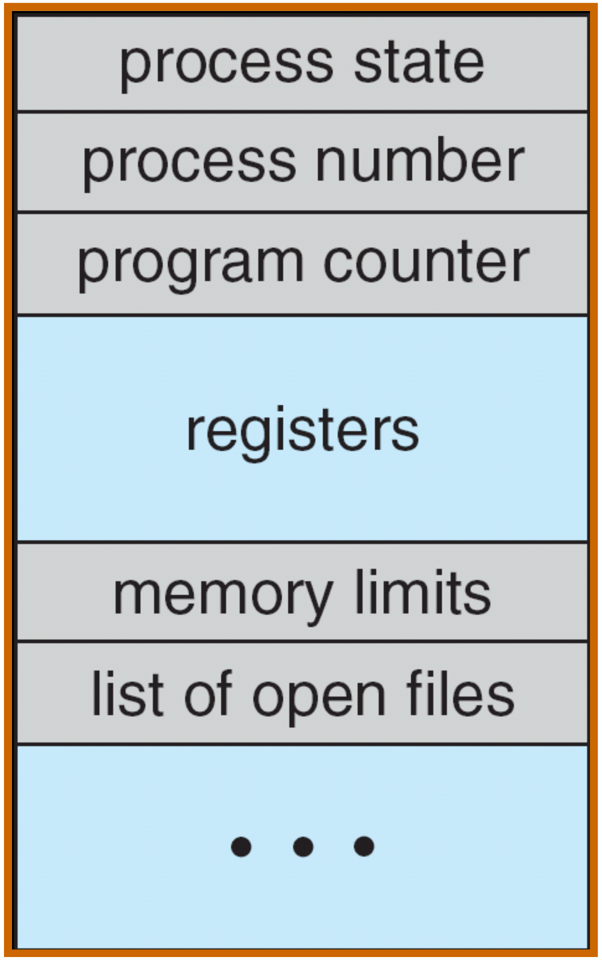


Process Concept – The process

Process Concept – Process State



Process Concept – Process Control Block



Process Concept – Process Control Block

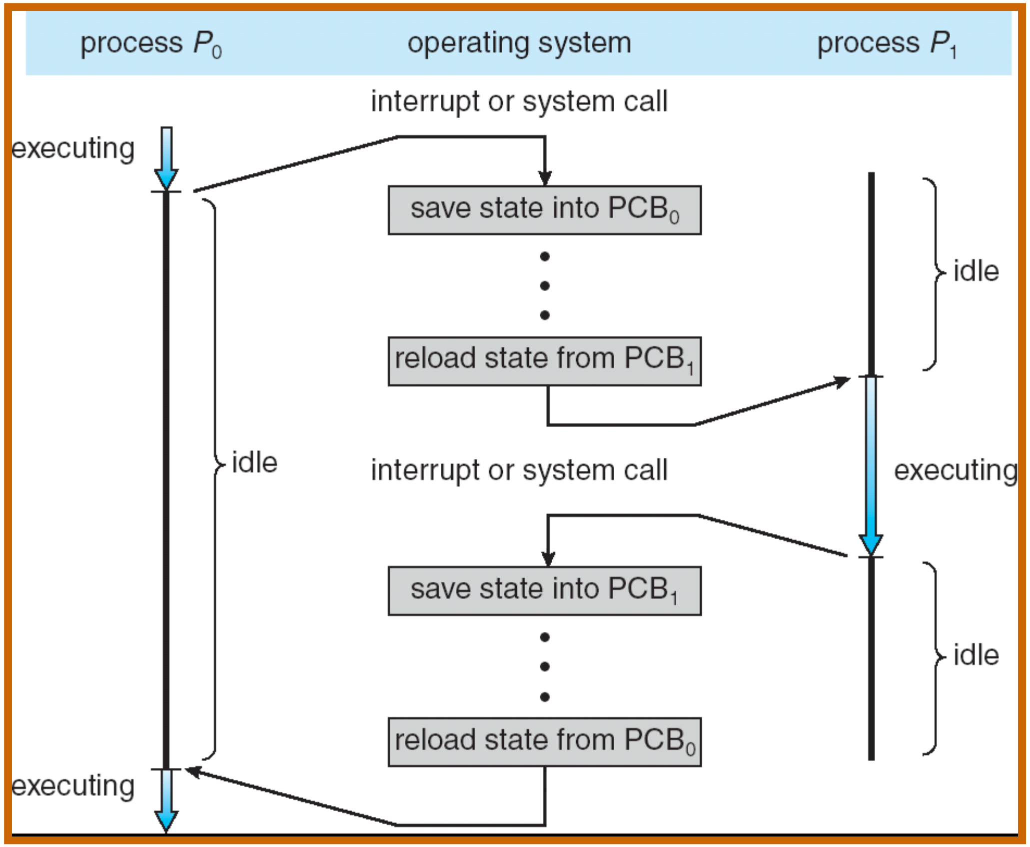
Information associated with each process

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

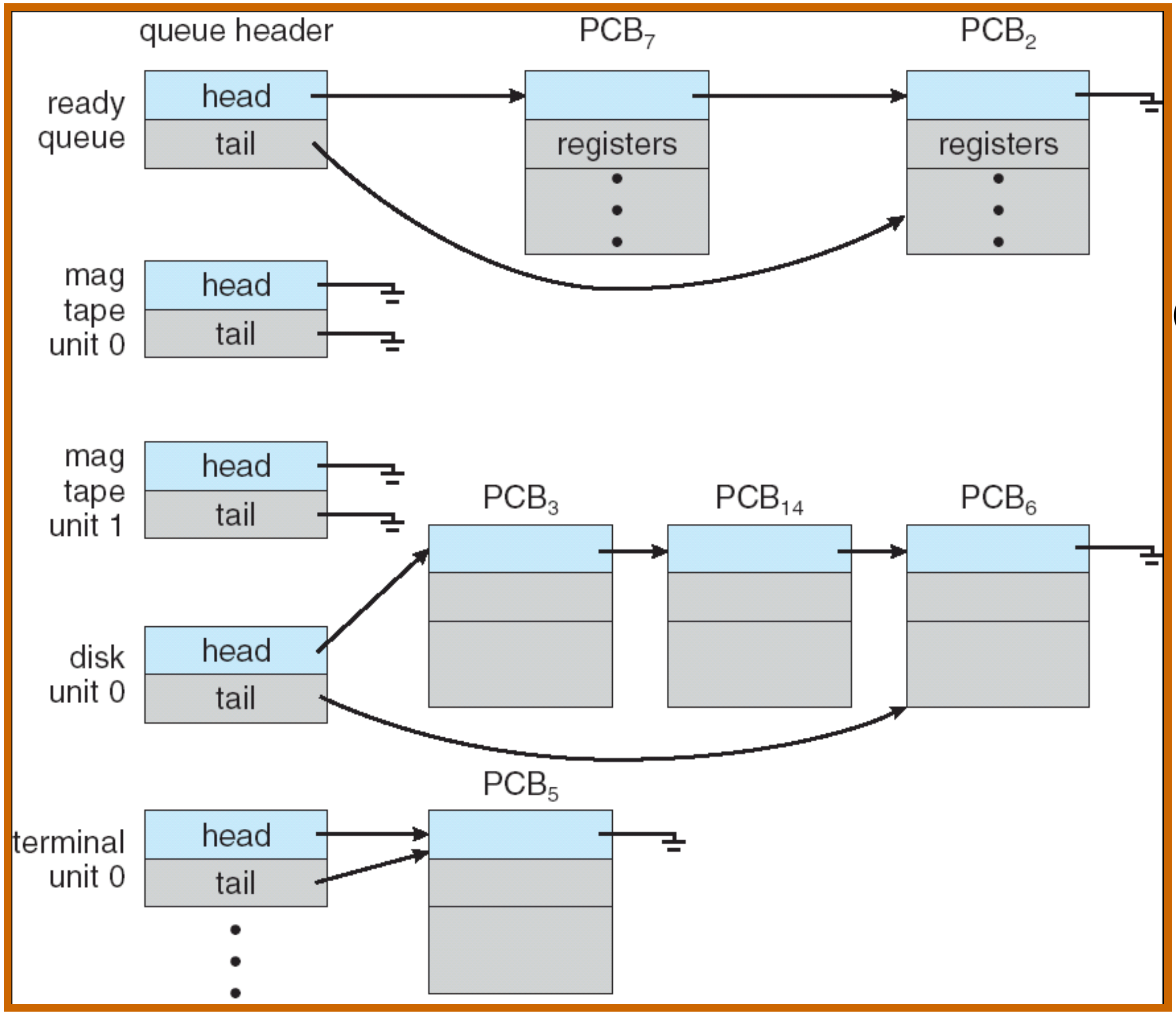
Process Concept – Process Control Block

- **Threads**
- Extended concept - processes have multiple threads of execution – to perform more than 1 task at a time
- PCB is expanded to include information for each thread

Process Concept – CPU Switch from process to process

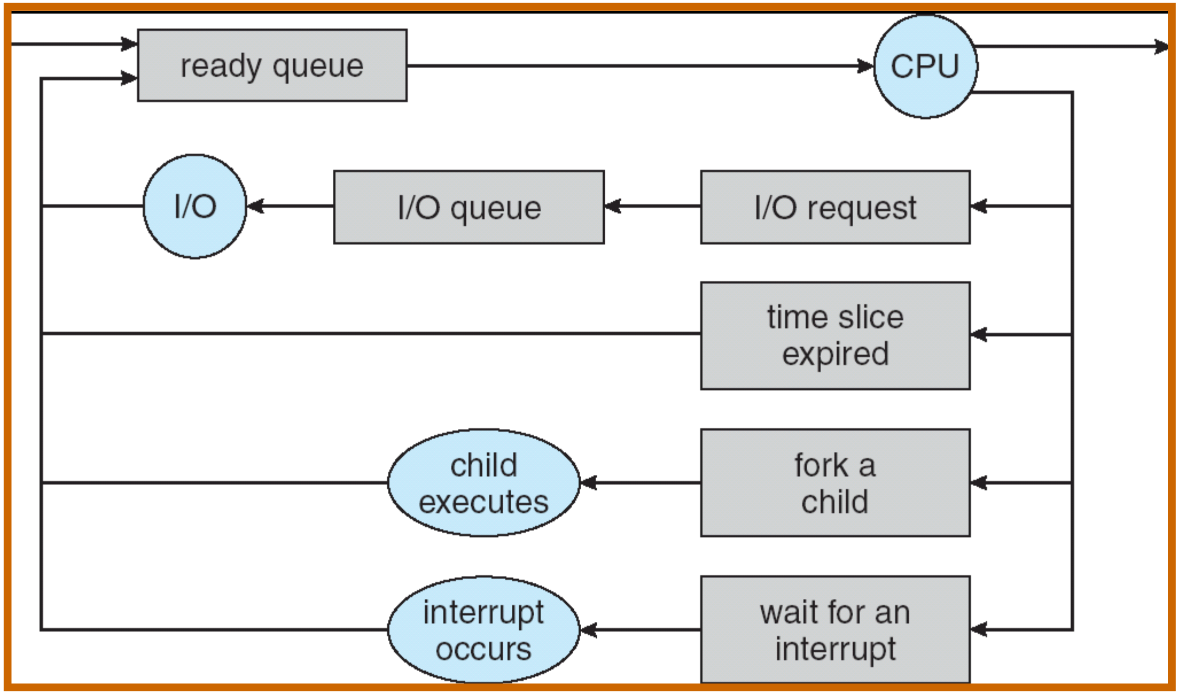


- **Job Queue, Ready Queue, Device Queue**



Process Scheduling – Scheduling Queues

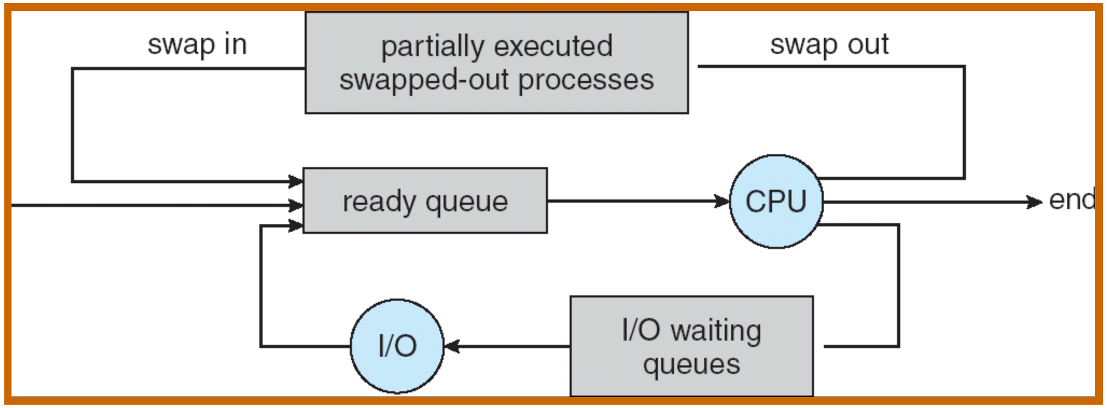
Process Scheduling – Representation of process Scheduling



Process Scheduling – Schedulers

- **Long term Scheduler –** degree of multi programming
- **Short term Scheduler** – Invoked frequently
- Medium Term Scheduler – remove process from memory (swapping)
- **I/O Bound process** – short CPU Bursts
- **CPU Bound Process** – long CPU Bursts

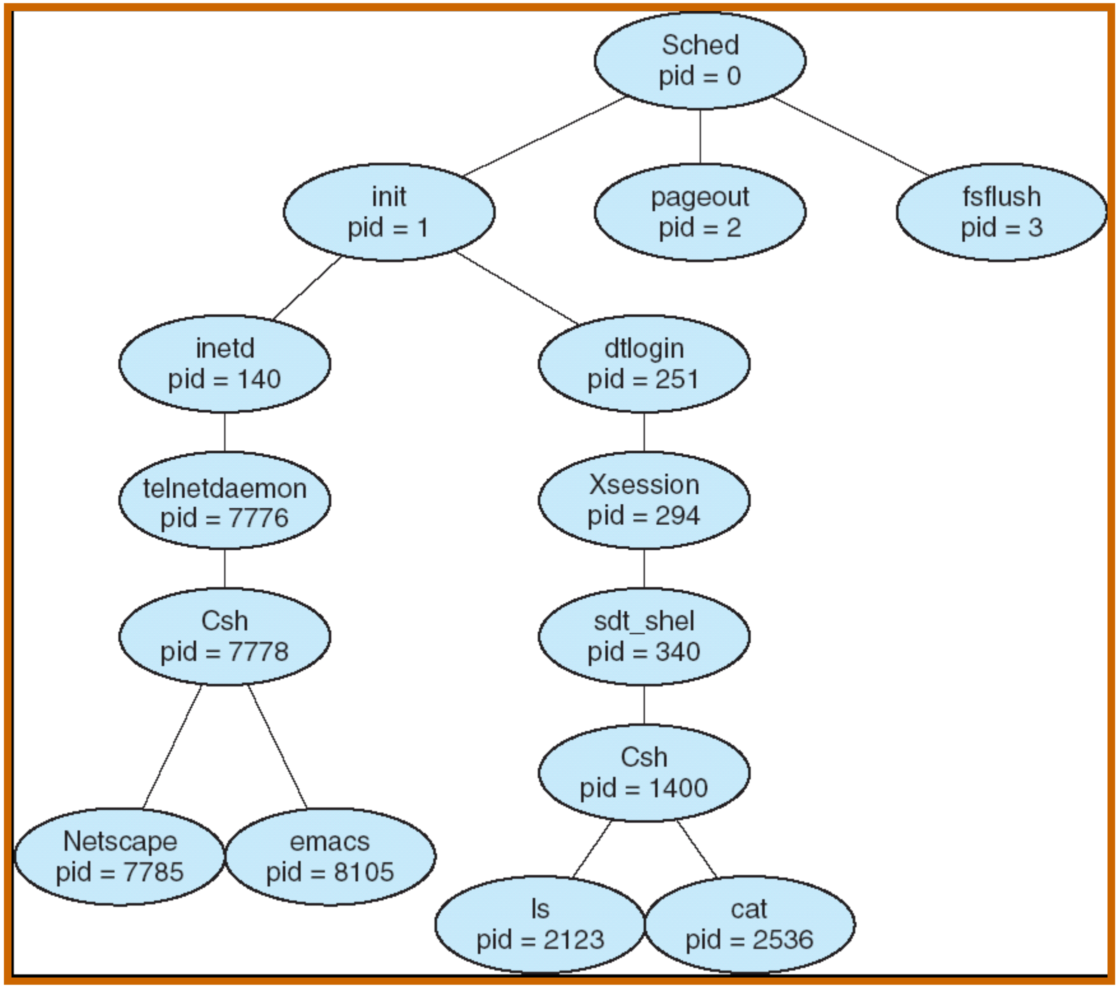
Process Scheduling – Medium Time Schedulers



Process Scheduling – Context Switch

- Switching the CPU to another process requires state save of current process and state restore of new process – Context Switch
- Switch time varies from system to system – dependent on hardware support(memory speed, registers)
- Context switch time – overhead

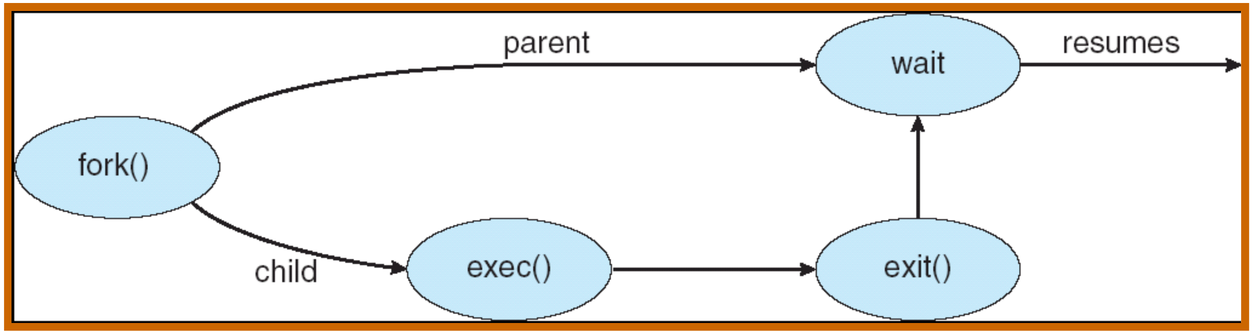
- **A tree of processes on a typical Solaris**



Operation on Processes– Process Creation

Operation on Processes– Process Creation

- Parent and child process
- Resource sharing all, subset or no resource
- Parent and child execute concurrently or wait
- Fork()
- Exec()
- Exit()



Operation on Processes– Process Creation

```
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) { /* 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);
      }
}
```

Linux ystem calls

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Operation on Processes–Process Creation

- Parent process -> create children processes -> 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– Process Creation

- **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

Operation on Processes– Process Termination

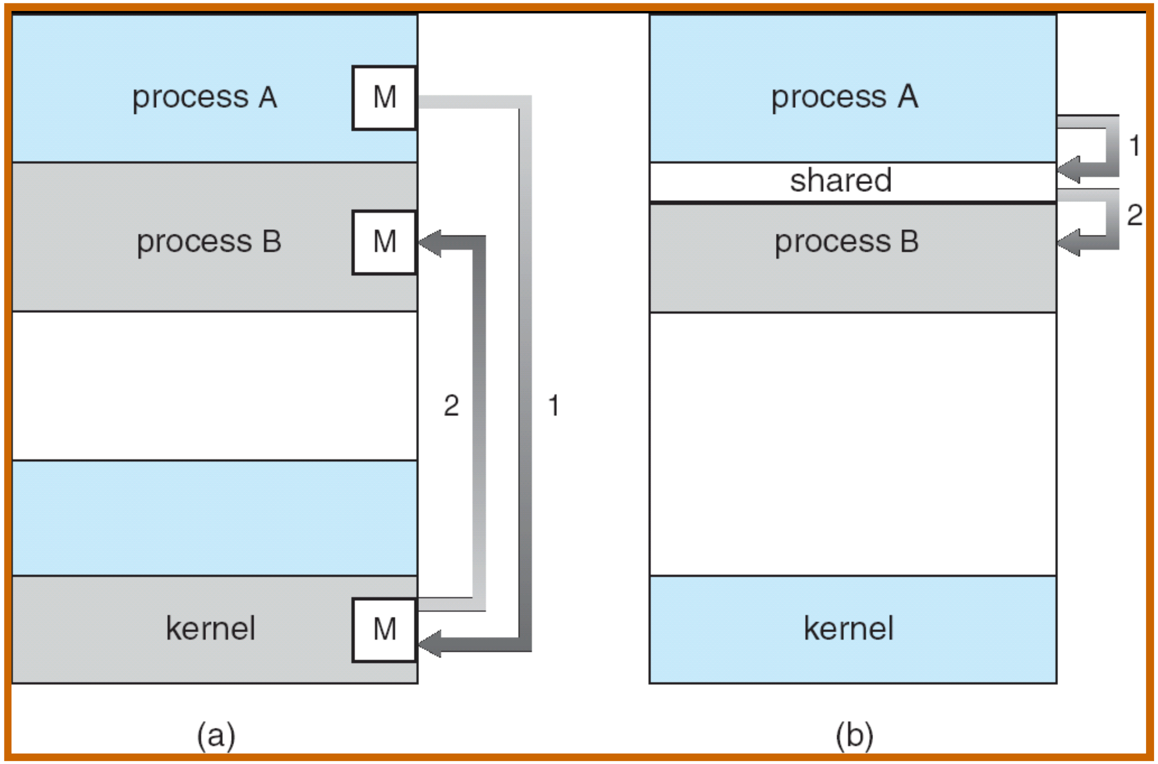
- **Exit**() system call
- Output data from child to parent (via **wait**)
- Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
- If parent is exiting child can continue or terminate (cascading termination)

Operation on Processes– Interprocess Communication

- **Independent** process no affect on other process
- **Cooperating** process can affect other process

- **Advantages of process cooperation**
  - **Information sharing** - shared file
  - **Computation speed-up** – subtasks executing in parallel
  - **Modularity** – system functions into separate processes or threads
  - **Convenience** – individual user working on many tasks at the same time

Operation on Processes– Interprocess Communication



- a) Message Passing b) Shared Memory

Operation on Processes– Interprocess Communication

- **Message Passing**
  - Useful for exchanging smaller amounts of data
  - No conflicts need to be avoided
  - Easier to implement even in inter computer communication
  - Typically implemented using system calls – kernel intervention
- **Shared Memory**
  - Faster – system calls are required only to establish shared memory regions

Operation on Processes– Interprocess Communication (Shared memory)

- **Eg: Producer Consumer Problem**

- Producer produces items, consumer consumes item
- Unbounded Buffer
- Bounded Buffer

- Shared buffer – a circular array with two logical pointers in and out
  - *#define BUFFER_SIZE 10*
  - *typedef struct {*
  - *. . .*
  - *} item;*
  - *item buffer[BUFFER_SIZE];*
  - *int in = 0; //next free position*
  - *int out = 0; //first full position*

Operation on Processes– Interprocess Communication (Shared Memory)

Producer Process

```
while (true) {
   /* Produce an item */

      while ((((in + 1) % BUFFER SIZE count)  ==

                  out)

         ;   /* do nothing -- no free buffers */

         buffer[in] = item;

         in = (in + 1) % BUFFER SIZE;

   }
```

Consumer Process

```
    while (true) {

        while (in == out)

              ; // do nothing -- nothing to consume


         // remove an item from the buffer

         item = buffer[out];

         out = (out + 1) % BUFFER SIZE;

    return item;

   }
```

Operation on Processes– Interprocess Communication (Message passing Systems)

- 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 of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Operation on Processes– Interprocess Communication – Message Passing

- **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?

Operation on Processes–Interprocess Communication (Message Passing Systems)

- Direct or Indirect Communication (naming)
- Synchronization
- Buffering

Operation on Processes– Message Passing Systems – 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
  - 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

Operation on Processes– Message Passing Systems – Indirect Communication

- Primitives are defined as:

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

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

- 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

Operation on Processes– Message Passing Systems – Indirect Communication

- 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.

Operation on Processes– Message Passing Systems – Indirect Communication

- Mailbox can be owned by a process or OS

- If process – then the receiver is known
- If OS – then provide mechanism

- Create a new mailbox
- Send or receive message through mailbox
- Delete the mailbox

Operation on Processes– Message Passing Systems – 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

Operation on Processes– Message Passing Systems – 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)

3. **Bounded capacity** – finite length of *n* messages
   Sender must wait if link full

3. **Unbounded capacity** – infinite length
Sender never waits