



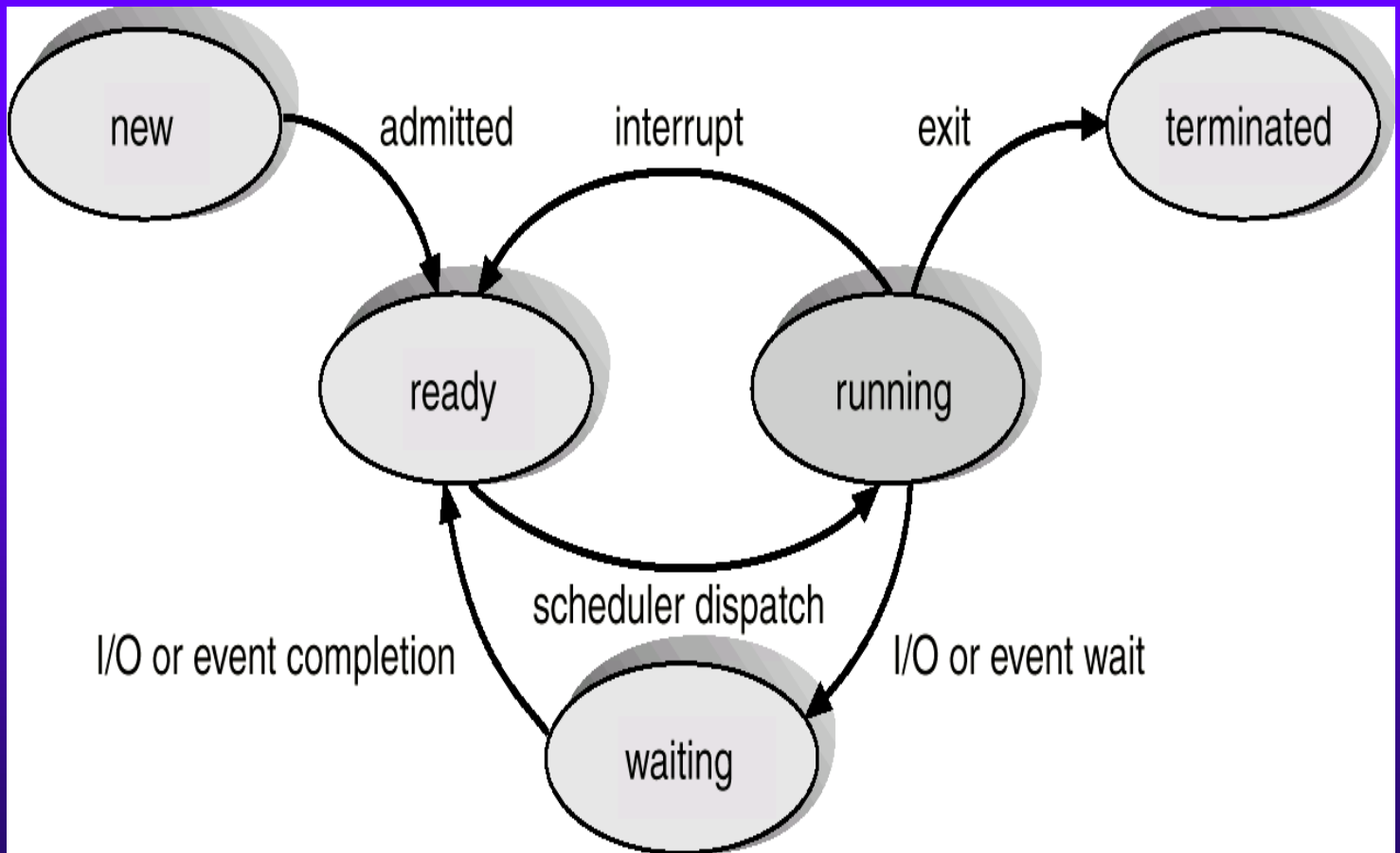
PROCESSES

Date 24/01/2004

Process concept

- Process is a program in execution. The program is the passive entity where as the process is the active entity.
- process execution must progress in sequential fashion.
- A process includes:
 - program counter,stack,data section etc...
- 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 process.
 - terminated: The process has finished execution.

Diagram of Process State



Process Control Block (PCB)

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
• • •	

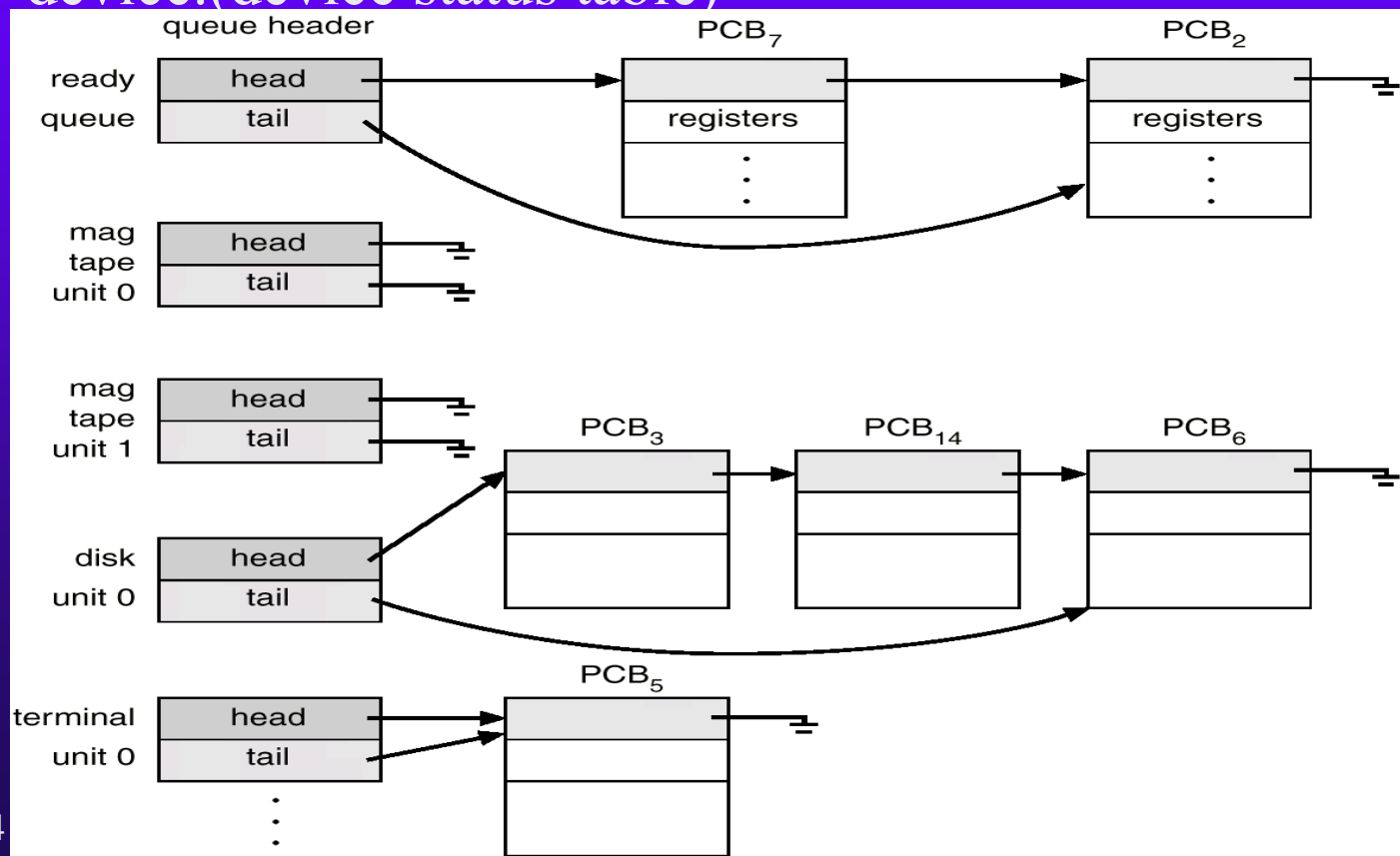
Process Control Block (PCB)

Information associated with each process.

- Process state
 - The State may be New,Ready,Running,Waiting,Halted...
- Program counter
 - Indicates the address of the next instruction to be executed.
- CPU registers
 - Various kinds of registers like Accumulator, index registers ...
- CPU scheduling information
 - This information includes process priority, pointers to scheduling queues,and any other scheduling parameters
- Memory-management information
 - This information includes value of limit and base register etc
- Accounting information
 - CPU & real time used, time limits, account no., job or process no..
- I/O status information
 - List of I/O devices allocated to this process (no of open files etc)

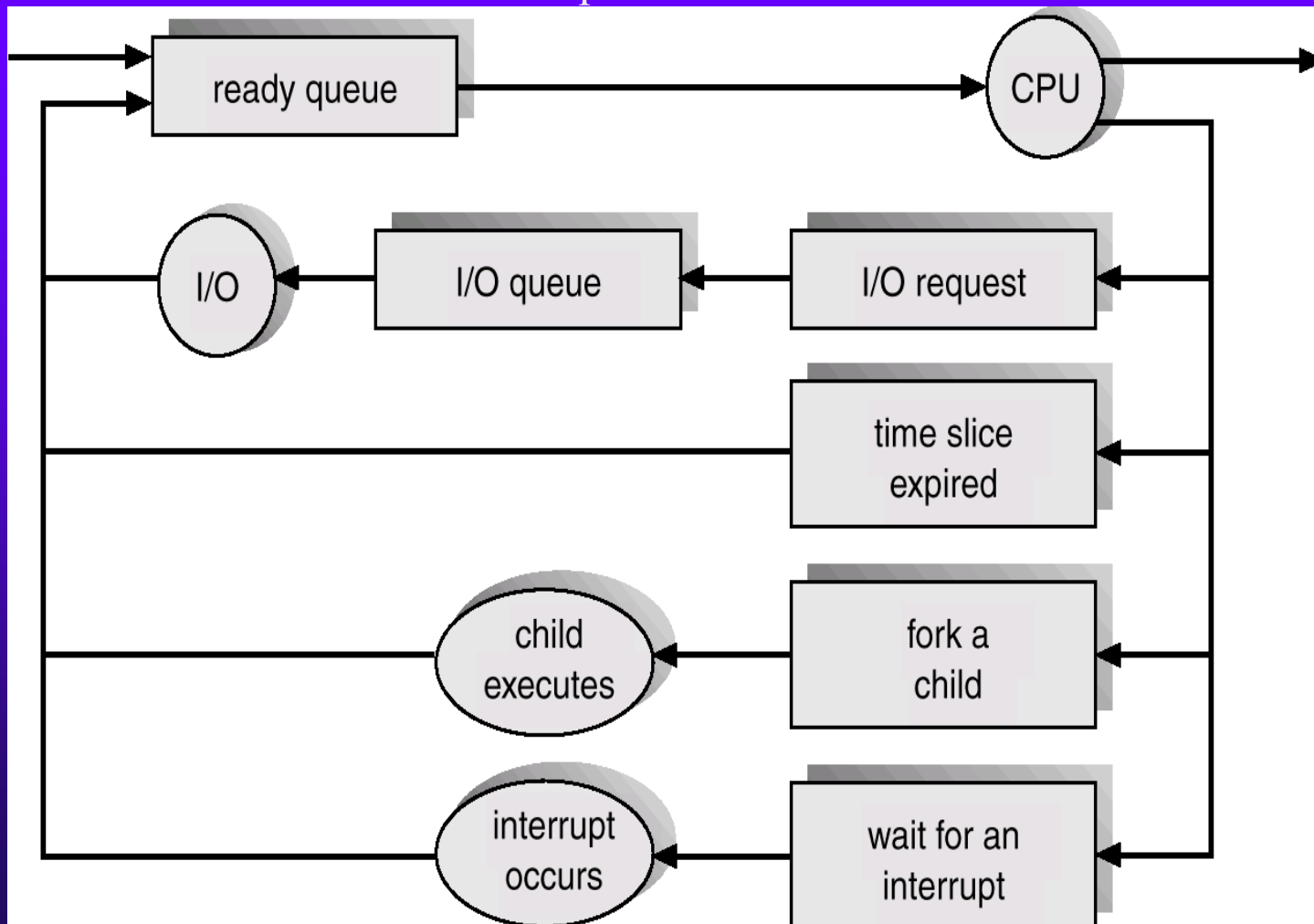
Process Scheduling

- 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 I/O device. (device status table)



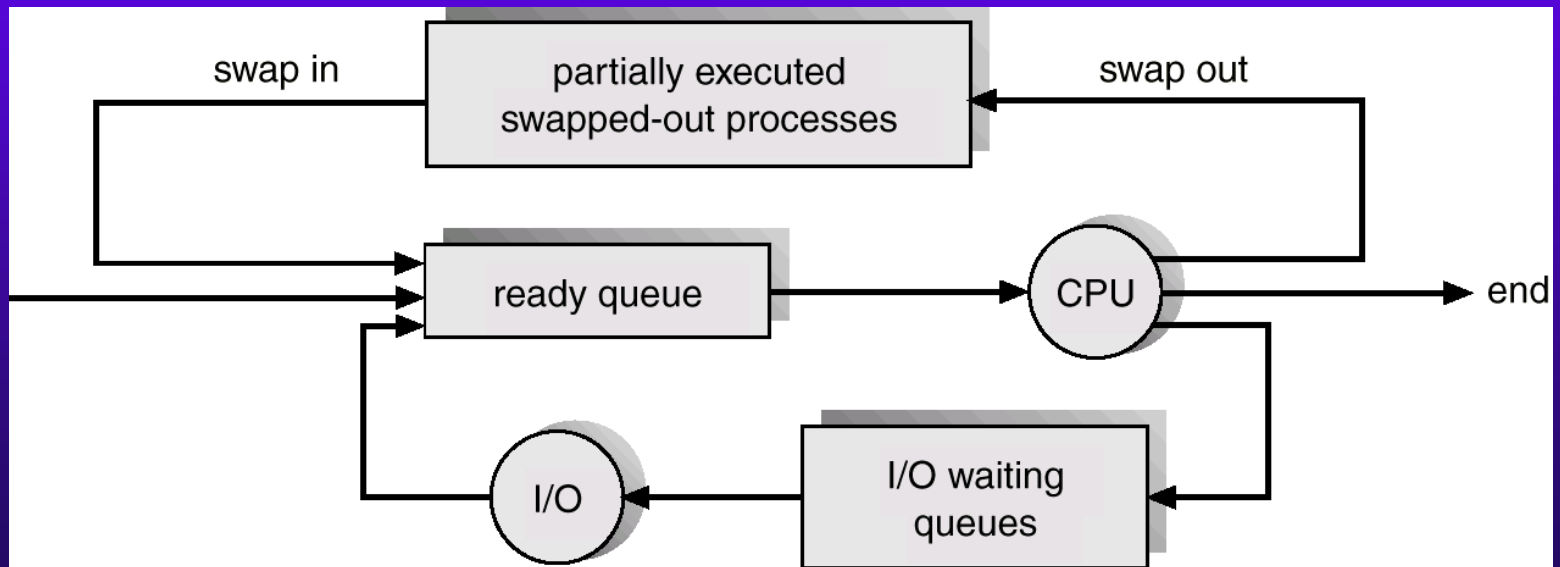
– Common representation of process scheduling is queueing diagram

- The rectangular boxes represents a queue(2 types of queues – ready queue and device queue) and circle represents the resources that serve the queue.



Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.
- Mid term scheduler



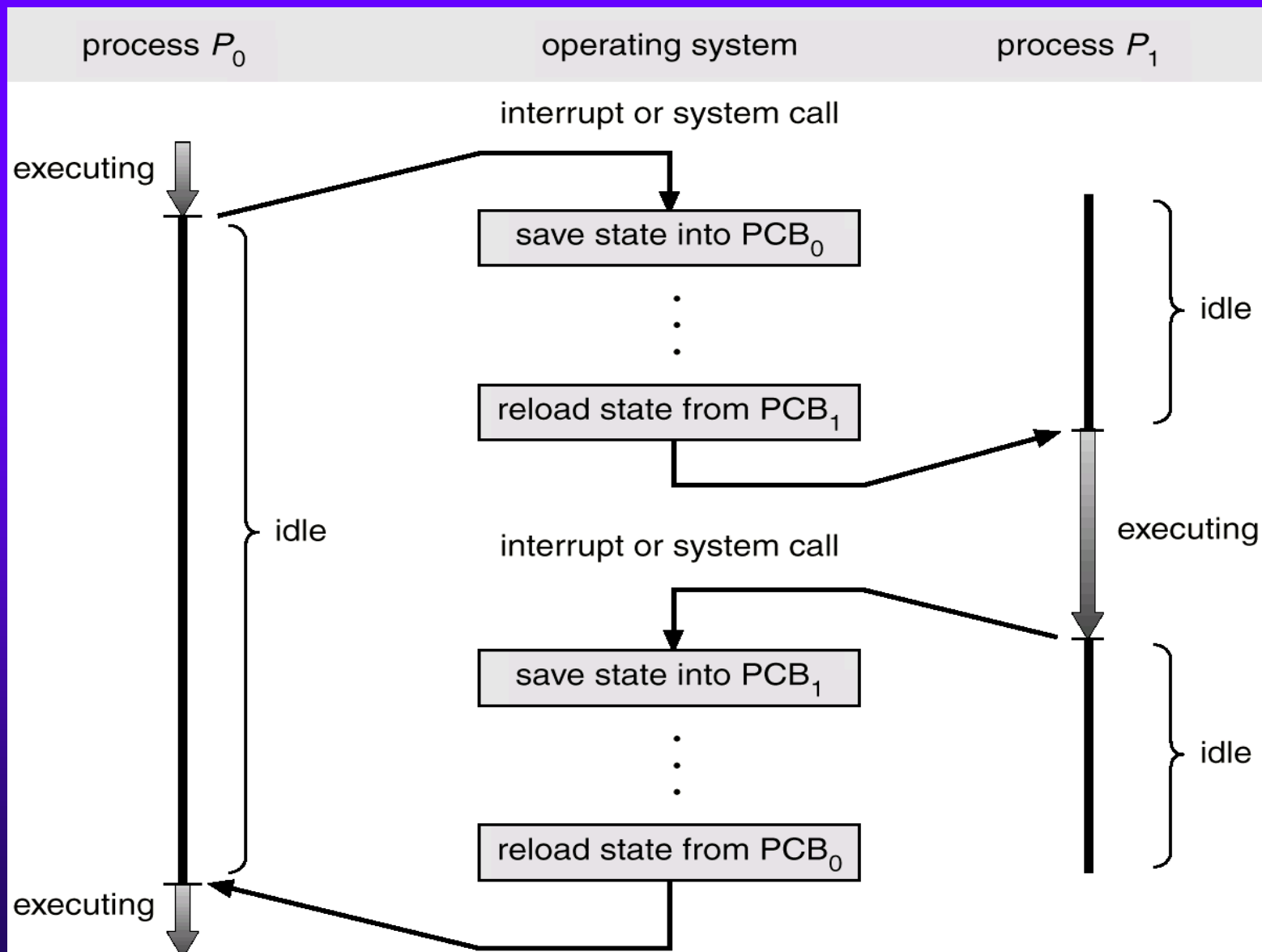


- Short-term scheduler is invoked very frequently (milliseconds) \Rightarrow (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) \Rightarrow (may be slow).
- The long-term scheduler controls the *degree of multiprogramming*.
- Processes can be described as either:
 - *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.
- The mid-term scheduler reduces the *degree of multiprogramming*.

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.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

CPU Switch From Process to Process



Process Creation

- Parent process creates 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.
- 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.

Example program

```
#include<stdio.h>

Void main ( int argc, char *argv[] )
{
    pid = fork ( );          // fork another process
    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
        wait ( NULL );      // Parent will wait for the child to complete
        printf ( “ Child Complete” );
        exit ( 0 );
    }
}
```

Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
 - 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.




Cooperating Processes

- ◆ *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
 - Computation speed-up
 - Modularity
 - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process. To allow producer and consumer process to run concurrently we must have a buffer. The producer and consumer must be synchronized.
 - *unbounded-buffer* places no practical limit on the size of the buffer.
 - *bounded-buffer* assumes that there is a fixed buffer size.

Program for producer consumer



```
//Shared data
#define BUFFER_SIZE 10
Typedef struct {
    .....
    .....
} item;
Item buffer[ BUFFER_SIZE ];
Int in = 0;
Int out = 0;

                Producer
Do {
    //Produce an item in nextProduced
    while ( ( ( in + 1 ) % BUFFER_SIZE ) == out );           // Do nothing
    buffer [ in ] = nextProduced;
    in = ( in + 1 ) % BUFFER_SIZE;
} while ( 1 );
```



consumer

```
Do {  
    while ( in == out );    // do nothing;  
    nextConsumed = buffer [ out ];  
    out = ( out + 1 ) % BUFFER_SIZE;  
        // Consume the item in nextConsumed  
}
```

nextProduced and nextConsumed are the local variables used in Producer and consumer code respectively.



Inter process Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- 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)

Various physical and logical implementation

- ◆ Physical implementation of link
 - Shared memory, hardware bus, networking
- ◆ Methods for Logically implementing a link & send and receive operation
 - Direct or indirect communication
 - Symmetric or asymmetric communication
 - Automatic or explicit buffering
 - Send by copy or send by reference
 - Fixed sized or variable sized messages

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 symmetric 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.

◆ Properties of asymmetric communication link

- Only the sender names the recipient;the recipient is not required to name the sender.
- The disadvantage (symmetric and asymmetric) is the limited modularity of the resulting process definition.

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.
- Operations(mailbox owned either by OS or by the user)
 - create a new mailbox
 - send and receive messages through mailbox
 - destroy a mailbox

Synchronization

- Message passing (send and receive) may be either blocking or non blocking also known as synchronous & asynchronous
 - Blocking send
 - Sending process is blocked until the message is received by the receiving process.
 - Non Blocking send
 - Sending process sends message and resumes operation.
 - Blocking receive
 - Receiving process blocks until message is available.
 - Non Blocking receive
 - Receiving process retrieves either 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).
sometimes referred to as a message system with no buffering or automatic buffering.
2. Bounded capacity – finite length of n messages
Sender must wait if link full.
3. Unbounded capacity – infinite length
Sender never waits.

◆ Mach OS IPC

- Most communication (including most of the system calls) is carried out by messages.
- With each new task 2 new (kernel & notify) special mailboxes are also created.
- Maximum size of message queue by default is 8.
- Multiple messages from the same sender are queued in FIFO order but doesn't guaranty an absolute order.



◆ Windows 2000

- Message passing facility is called Local Procedure Call (LPC)
- Like Mach windows also uses port object to establish and maintain connection.
- Steps of communication
 - Client opens a handle to the subsystem's connection port object
 - Client sends a connection request
 - Server creates 2 private communication ports, and returns the handle to one of them to the client.
 - The client and server use corresponding port handle to send messages or callbacks and to listen for replies.

