OPERATING SYSTEMS

PROCESSES

Definitions

Processes

- The OS has to manage the concurrent execution of programs
- The execution of a program on a computer is called a process

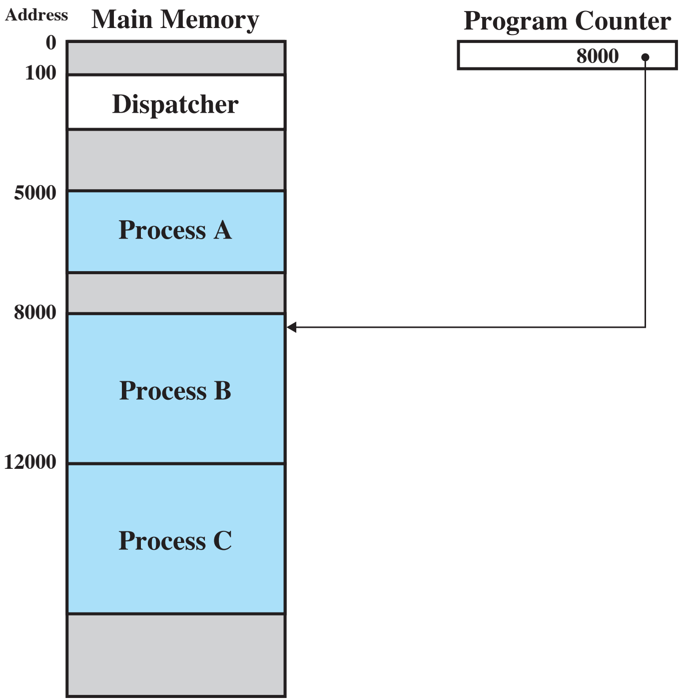
Terminology

- Old operating systems where designed to schedule batch jobs
- When multiprogramming was introduced, as well as time-sharing, then the concept of process was introduced.
- However, we still have some algorithms used by the operating system that still have the word job in their name.

Informal definition of process

- A process represents a program in execution that is characterised by
  - the sequence of instructions to be executed
  - the CPU state (program counter, registers, etc.)
  - data, i.e., the program variables
  - return addresses related to function and procedure calls
- More than one process can be originated by the same program
  - e.g., two users executing the same program
- Process creation
  - console: type the name of the program and hit "return"
  - GUI: double click on the icon

Example Three processes



Example Execution traces

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

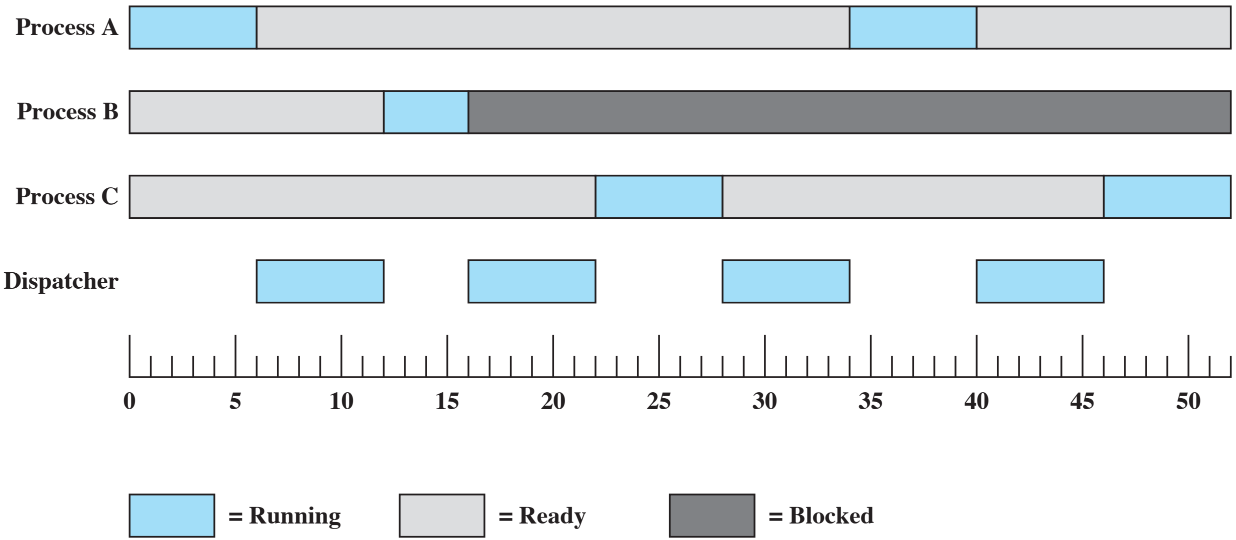
Example Sequence of execution

1	5000		27	12004
2	5001		28	12005
3	5002		-------------------- Timeout	
4	5003		29	100
5	5004		30	101
6	5005		31	102
-------------------- Timeout			32	103
7	100		33	104
8	101		34	105
9	102		35	5006
10	103		36	5007
11	104		37	5008
12	105		38	5009
13	8000		39	5010
14	8001		40	5011
15	8002		-------------------- Timeout	
16	8003		41	100
----------------I/O Request			42	101
17	100		43	102
18	101		44	103
19	102		45	104
20	103		46	105
21	104		47	12006
22	105		48	12007
23	12000		49	12008
24	12001		50	12009
25	12002		51	12010
26	12003		52	12011
			-------------------- Timeout	

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

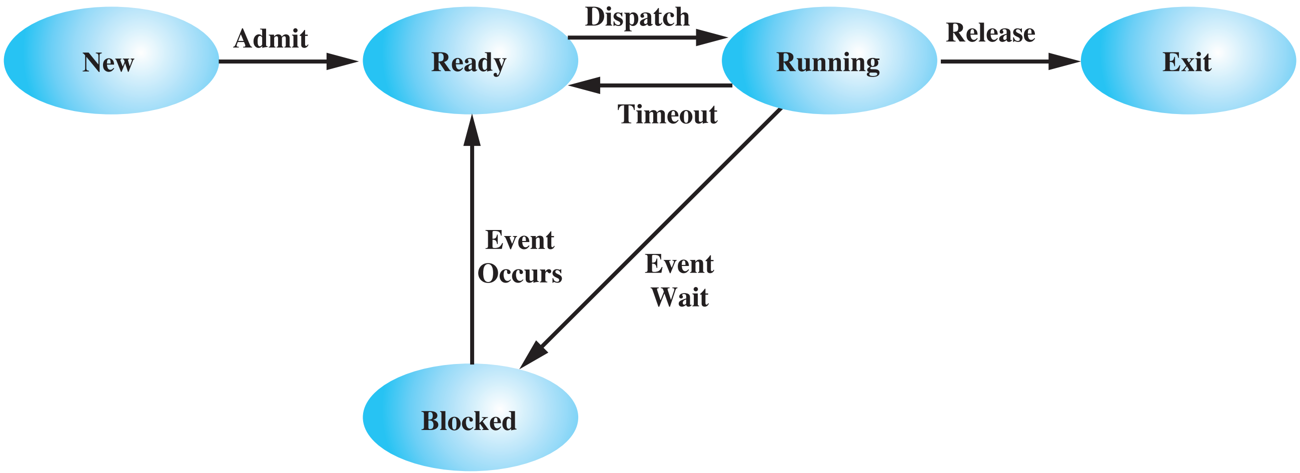
Example State transitions



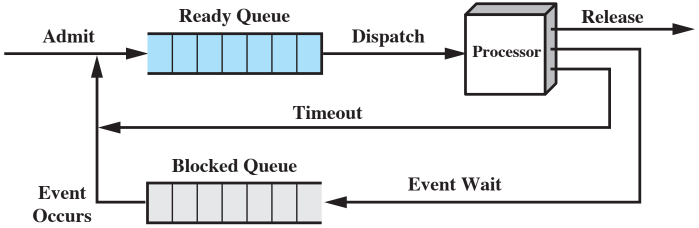
Process States

- A state diagram is used to describe the phases of execution of a process
  - the details differ from one system to another
- Five states
  - **New**
    the process is created, data structures are initialised
  - **Running**
    in uniprocessor system, only one process is running
  - **Ready**
    the process is ready but the CPU is already in use
  - **Blocked**
    the process is waiting for some event
  - **Exit**
    results are released and data structures are updated

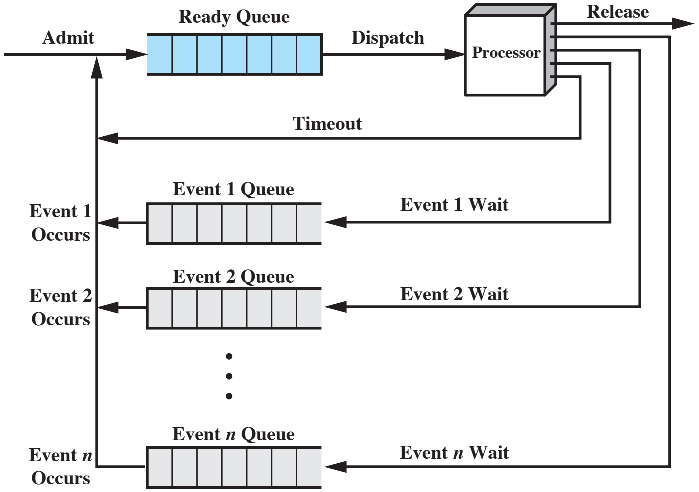
Five-State Process Model



Queues for process management

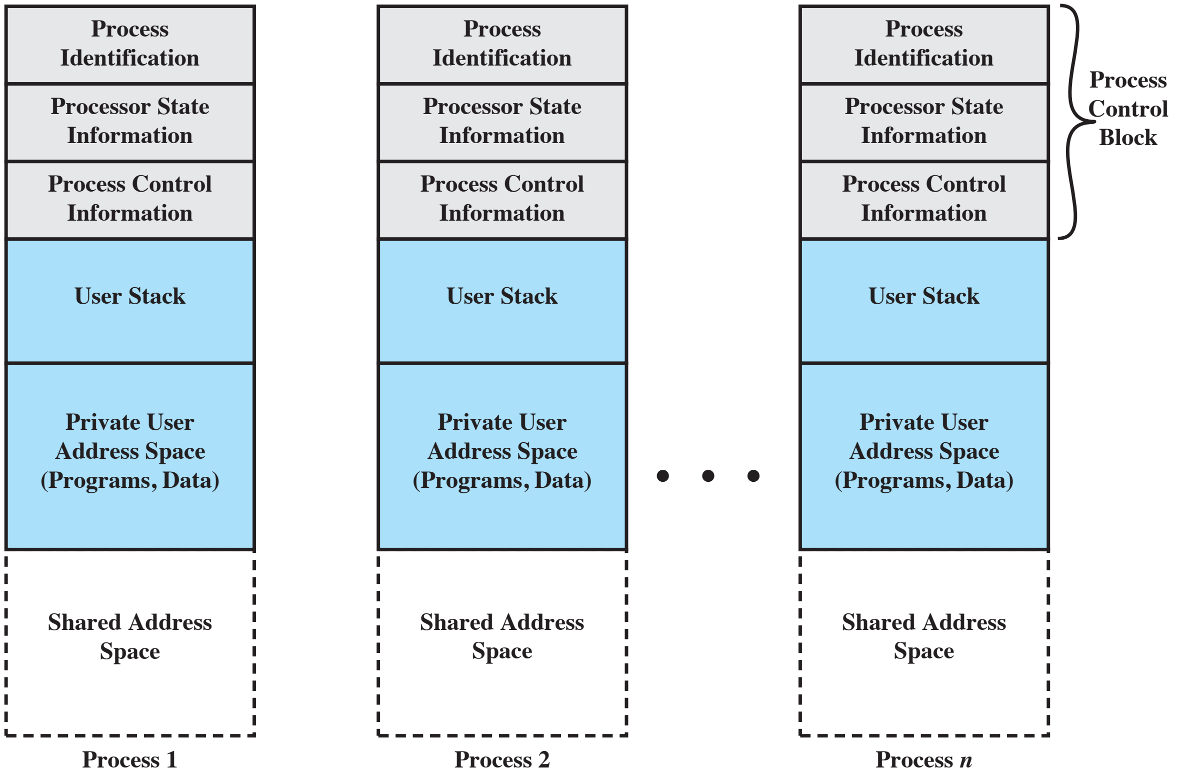


(a) Single blocked queue

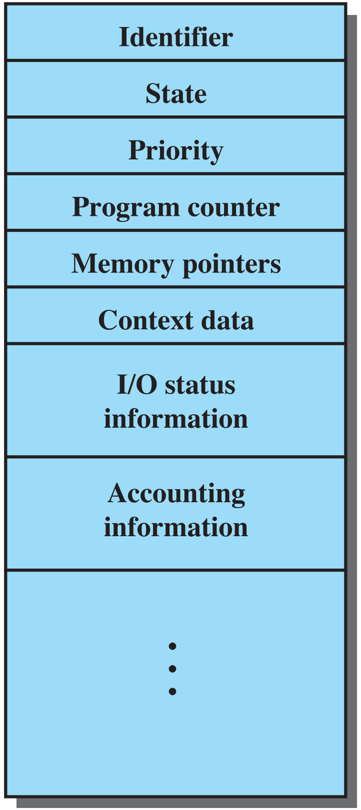


(b) Multiple blocked queues

Process image

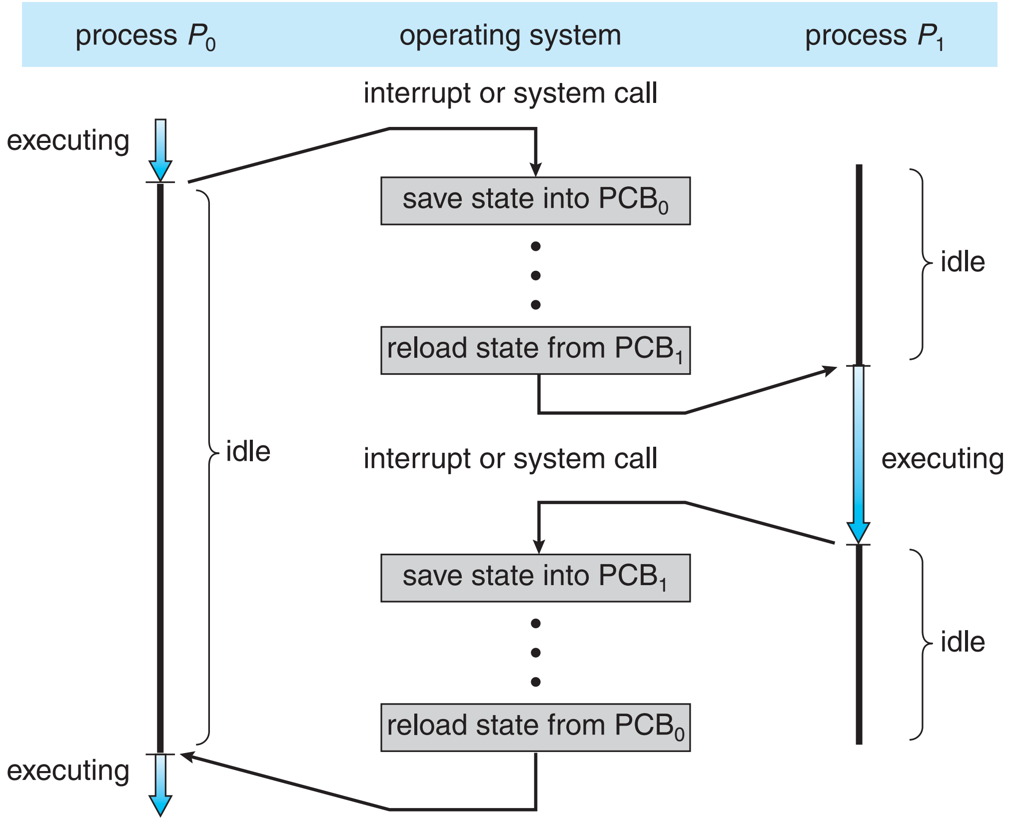


Process Control Block (PCB)



This component contains the information needed by the OS to identify the process and control its execution

PCB and process switch

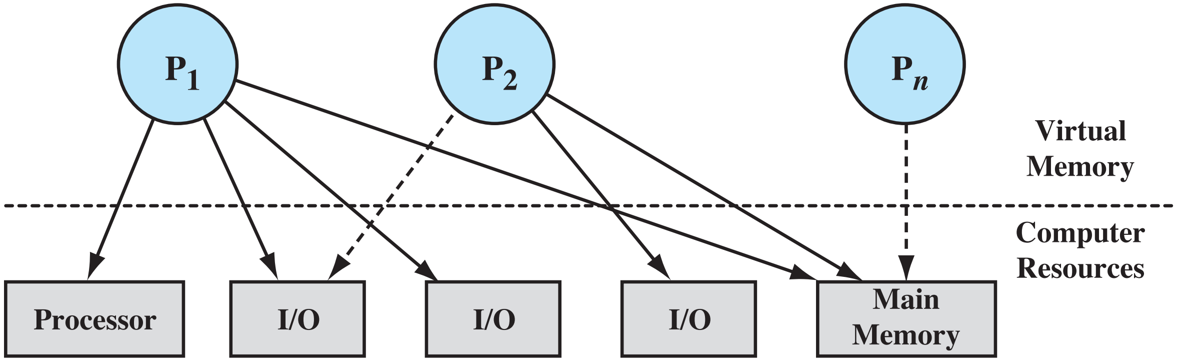


Thread

- The OS may allow different threads of execution within the same process space
  - e.g., the spell checker in a text editor
- The process structure and the process control mechanisms are modelled accordingly
- The vast majority of current OS are multithreaded

Process Description

Processes and system resources

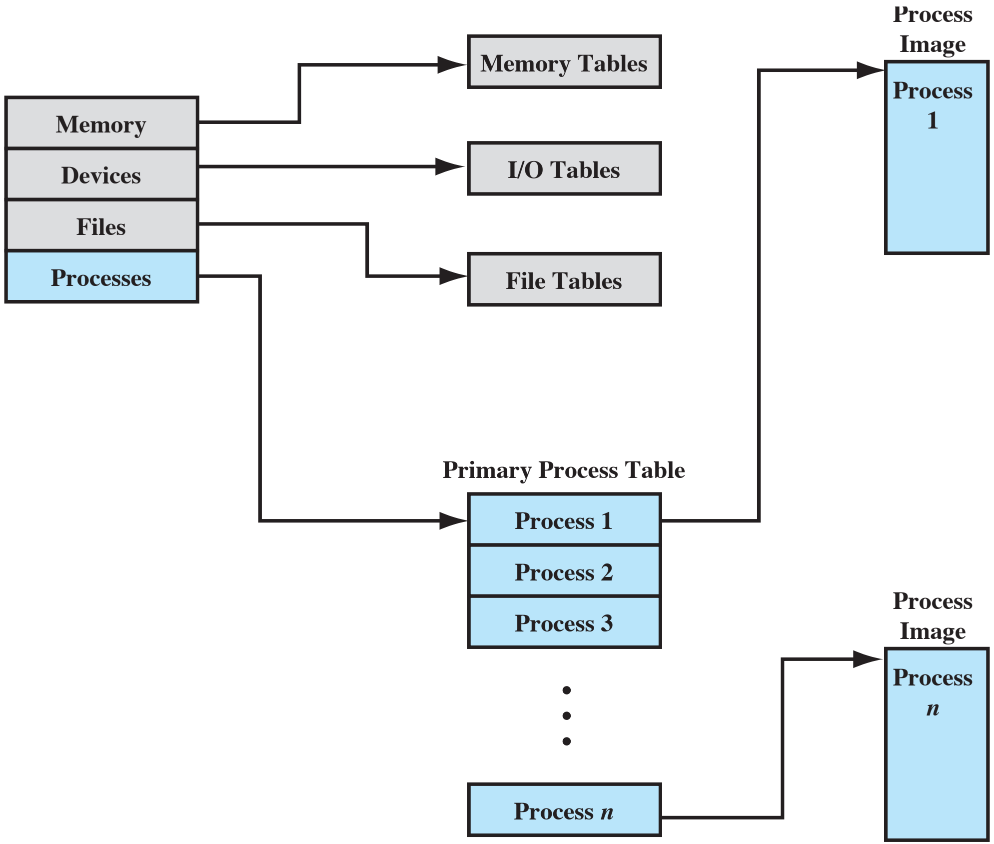


At a given time *t*

- One process has a number of resources allocated
- Each resource is allocated to 0, 1 or more processes

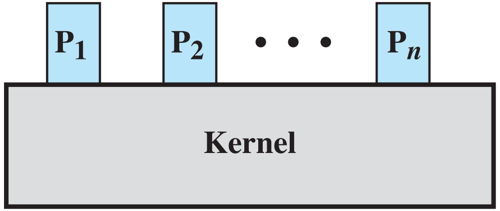
Resource allocation is controlled by the OS

OS Tables

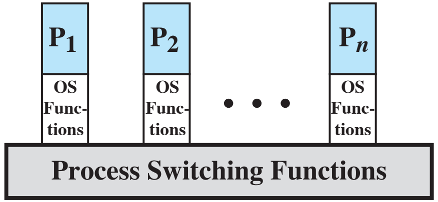


Memory
Devices
Files
Processes
Memory Tables
I/O Tables
File Tables
Primary Process Table
Process 1
Process 2
Process 3
Process *n*
Process Image
Process 1
Process Image
Process *n*

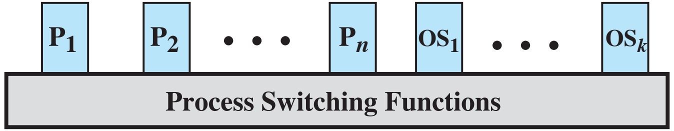
Execution of the OS



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Process Scheduling

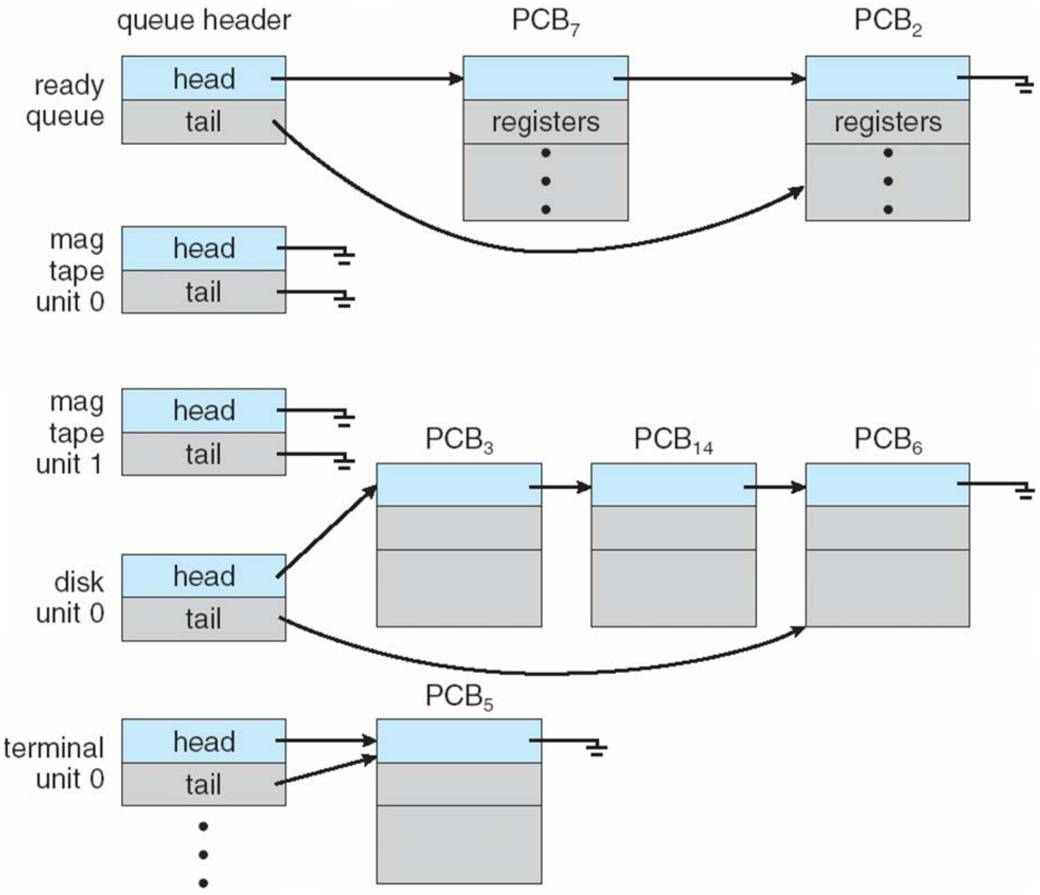
Goals

- Multiprogramming
  - Maximise the use of the CPU
- Time-sharing
  - The CPU is shared among different processes and users and the goal is to minimise the response time for each process and user

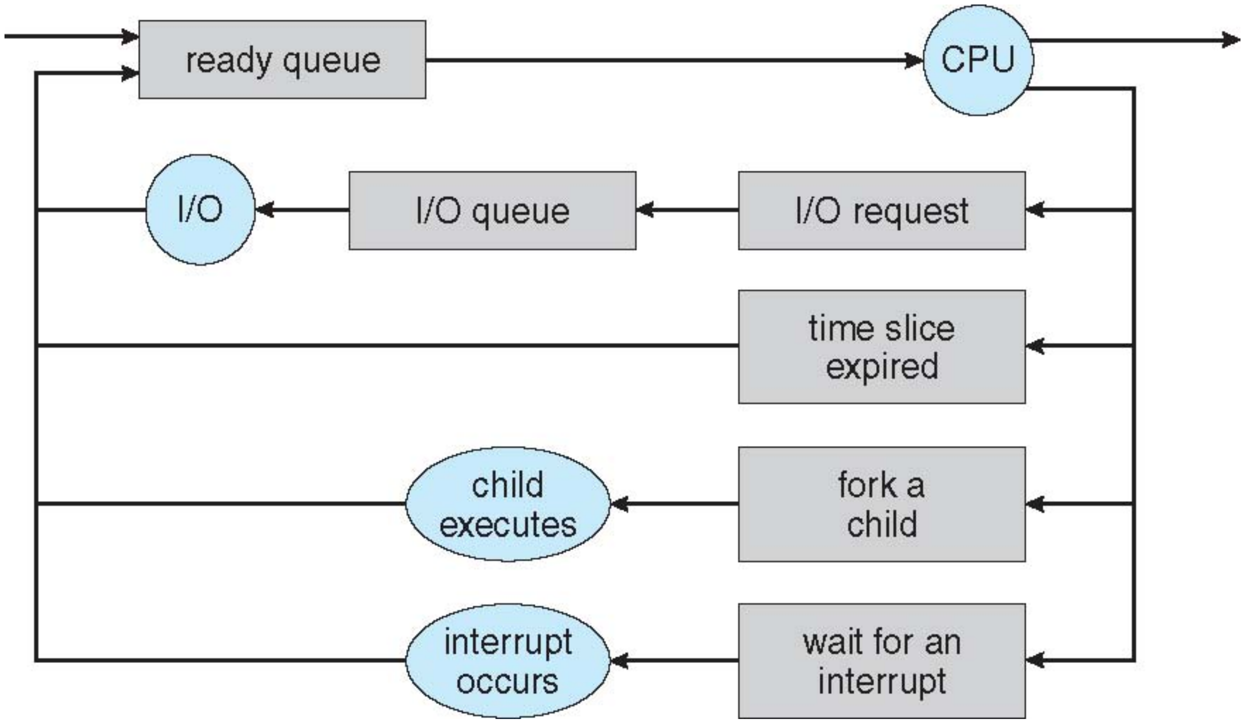
Scheduler

- This is one of the core component of the OS
  Each time that one processor (core) is idle, the scheduler selects one of the processes in the ready queue
  - different criteria can be used

Ready queues and I/O Device Queues



Queueing diagram



Short-term scheduling

- Also called the **dispatcher**
  - The task is to select the next process to be executed from the ready queue
  - The algorithm should be as fast as possible
    - the time required to the dispatcher to take a decision should be significantly smaller than the average CPU burst
      i.e., the time frame in which a process is in the running state
    - Process switch is a frequent operation, especially for time-sharing systems

Context switch

- This is the name given to the changes in the CPU registers when the OS interrupts the execution of process P_i and starts or resume the execution of process P_j
- The speed of the context switch depends on
  - the hardware architecture
    - e g., the CPU may contain different groups of registers, each group associated to one of the processes in execution
  - the size of the context
- When the OS takes control, there is no context switch but only a mode switch

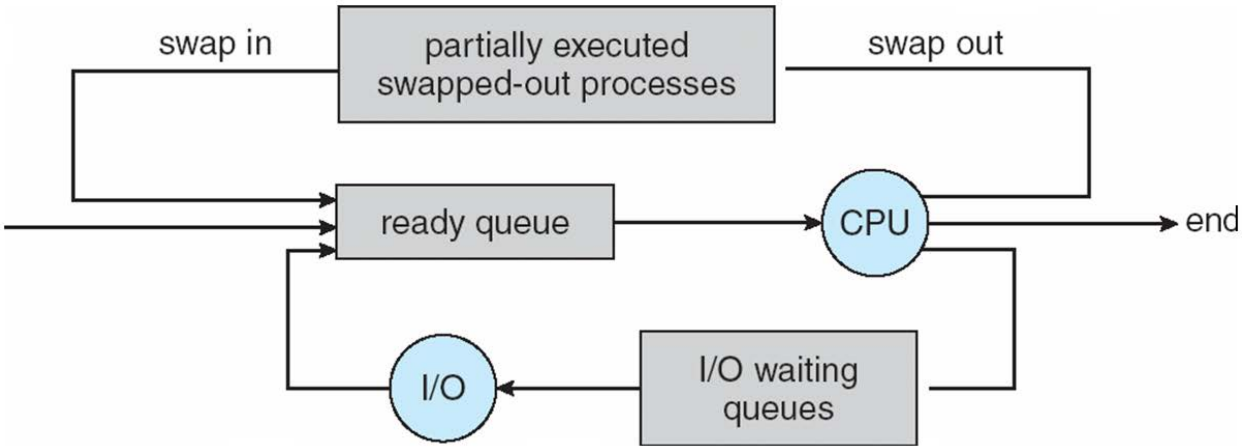
Long-term Scheduling

- Typical of batch systems
- It controls the degree of multiprogramming
  - when a process should be created
  - when a new process should join the ready queue
    e.g., only after the termination of one of the processes in execution
- UNIX and Windows does not have long-term scheduling functions

Long-term Scheduling

- It is not called with high frequency
  - The algorithm is not required to be fast
- The goal of the algorithm is to maximise the overall system usage through the concurrent execution of CPU bound and I/O bound processes
  - CPU bound processes: they have an intensive use of the CPU
  - I/O bound processes: they heavily use I/O

Medium-term Scheduling



- This is typical of time-sharing systems
- The degree of multiprogramming is controlled through swapping, i.e., moving some processes from the main memory to the hard disk

Operations on Processes

Process Creation

- UNIX model
  a new process is created by another process in execution
- The new process is called the child and the other process is called the parent
- Each process is identified by a numerical identifier
- Resources of the new process
  - allocated by the OS
  - some relations with the resources of the parent process

Parent-Child relationship

- The parent process can share the resources with the child process
  - memory locations, open files, communication channels, etc.
- Cooperation among processes
- The parent process can initialise the child process
- After the child is created
  - the parent continue executing until the OS switches to the child
  - the parent waits until the completion of the child processes
- Two alternatives for the image of the child process
  - a copy of the parent process
  - a new program

Process Termination

- Any process terminates with the `exit()` system call
  - all the resources are deallocated
  - a child process sends some data to the parent process
- The `exit()` system call may terminate any child process that is still executing
- It is possible to force termination
  - the administrator
  - a parent process can force the termination of any child processes
  - the OS

InterProcess Communication

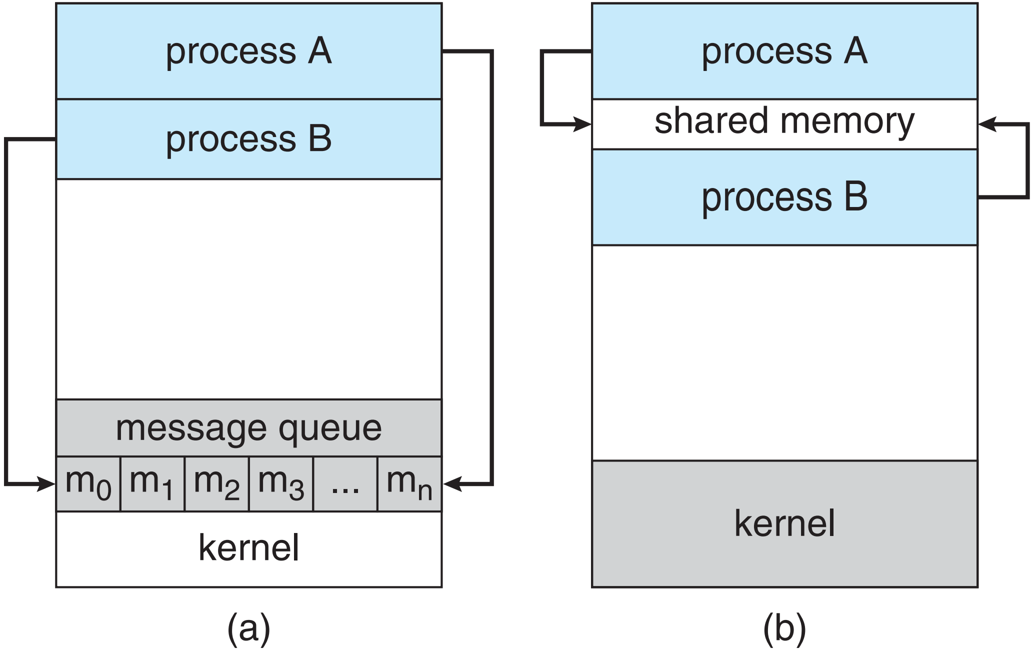
Motivations

- Process communication is used to
  - share information between cooperating processes
  - increase the speed of execution by distributing the computation on multiple processors or cores
  - exploit program modularity when different activities of the same program are implemented as concurrent processes or threads
  - make the use of the computer more convenient allowing the user to perform multiple concurrent activities

IPC InterProcess Communication

- This is the name of the group of system calls that implements process communication and synchronisation mechanisms
- Two main communication techniques
  - use of shared memory locations
    - the kernel is called for creating the area, then processes can communicate without calling the kernel
  - message passing
    - the kernel is called for the delivery

Communication models



message passing

shared memory

Shared memory
The producer-consumer problem

Typical paradigm for cooperating processes.

Some activities of the operating system are implemented according to this paradigm

Producer

- Produce some data and insert them in a buffer where the consumer has access to

Consumer

- Take the data from the buffer and use them

Shared Buffer

- Typically implemented as a circular array

Buffer of the producer-consumer problem

```
#define DIM_BUFFER 10

typedef struct {
  . . .
} item;

item buffer[DIM_BUFFER];
int in  = 0;
int out = 0;
```

Producer

```
item next_produced;
while (true) {
      /* produce an item in next produced */
      while (((in + 1) % BUFFER_SIZE) == out)
            ; /* do nothing */
      buffer[in] = next_produced;
      in = (in + 1) % BUFFER_SIZE;
}
```

The buffer can store `DIM_BUFFER – 1` items at most

Consumer

```
item next_consumed;
while (true) {
      while (in == out)
            ; /* do nothing */
      next_consumed = buffer[out];
      out = (out + 1) % BUFFER_SIZE;

      /* consume the item in next consumed */
}
```

Message Passing

- No need for a shared memory space
- Allows the communication between unrelated processes
- Based on two logical functions
  - `send(message)`
  - `receive(message)`
- Different communication modalities
  - direct or indirect
  - synchronous or asynchronous
  - automatic or explicit buffer management

Direct communication

- Between process pairs
- Each process must specify the process he wants to communicate to
  - **symmetric** communication
    both processes have to specify the other process
  - **asymmetric** communication
    only the sender specifies the other process
  - automatic communication channel provided by the OS

Indirect communication

- More flexibility w.r.t. direct communication mechanisms
- The OS implements ports and mailboxes
  - processes communicates by specifying the port
  - the port can be associated to one or more processes
  - two processes can use more than one port to communicate

Synchronization

- Synchronous (a.k.a. blocking) communication
  - **blocking send**

    the sender waits until the other process acknowledges the receipt
  - **blocking receive**

    the receiver waits until a message is sent
- Asynchronous (a.k.a. non-blocking) communication
  - **non-blocking send**

    the sender sends a message and continues
  - **non-blocking receive**

    the receiver either receives a valid message or a null value

Synchronization

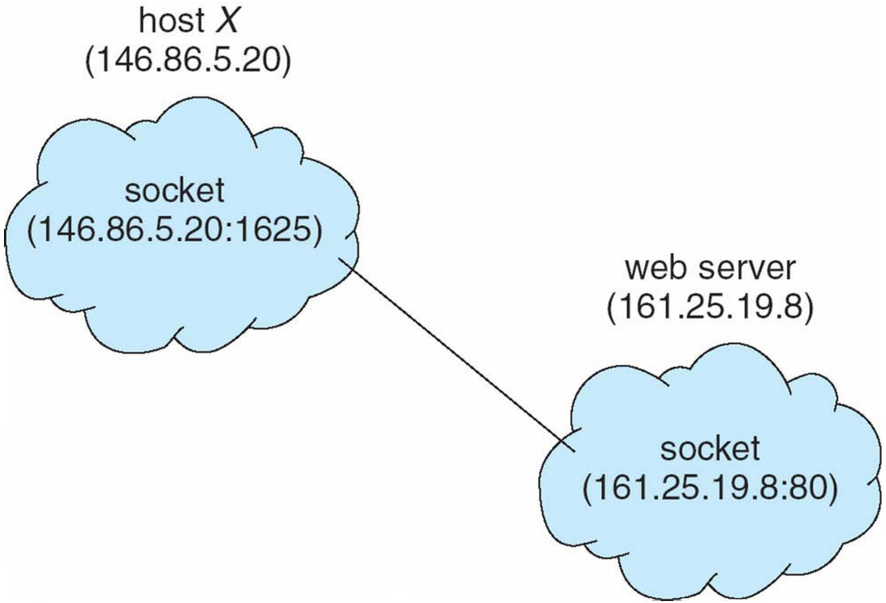
- Process communication can be implemented by any combination of synchronous and asynchronous send and receive
- A typical configuration is the **rendez-vouz**
  - both send and receive are blocking
  - it can be used to solve the producer-consumer problem

Message queues

- **Zero capacity** (*no buffering*)
  - The queue cannot contain messages to be delivered
- **Limited capacity** (*automatic buffering*)
  - The queue can contain up to N messages
    When the queue is full, the sender must wait
- **Unlimited capacity** (*automatic buffering*)
  - The queue has no limit in the number of messages

Client-server communication

Socket communication



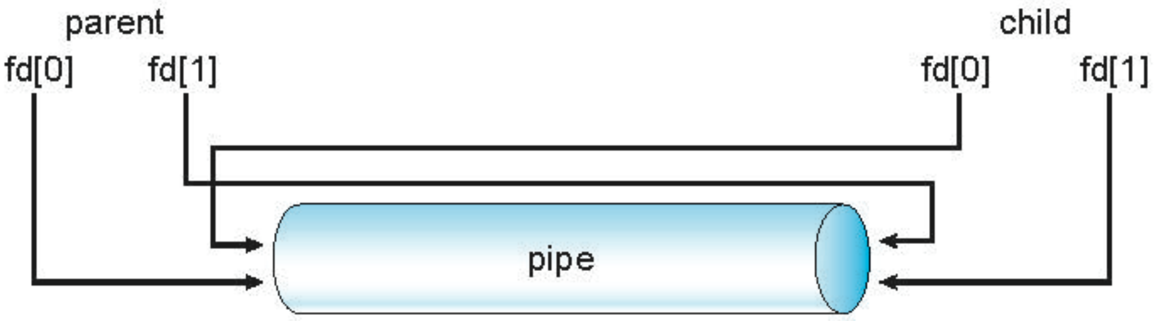
- Services are associated to ports that are identified by an integer
  - *http* port 80, *ftp* port 21, *ssh* port 22, etc.
- The client sends a request to one of the standard ports and waits for the reply to a port with number > 1024

RPC
Remote Procedure Call

- Client processes can call a procedure that is physically stored on a remote server
  - The client program includes a stub that allows the compiler to be unaware of the remote procedure
  - The stub is in charge to locate the server, and send the data to the remote procedure in the correct format

Pipe

- Communication channel between processes
  - **unnamed pipe** for processes in a parent/child relationship
  - **named pipe** any process pair



unnamed pipe