Process Type

→ Processes can be described as either:

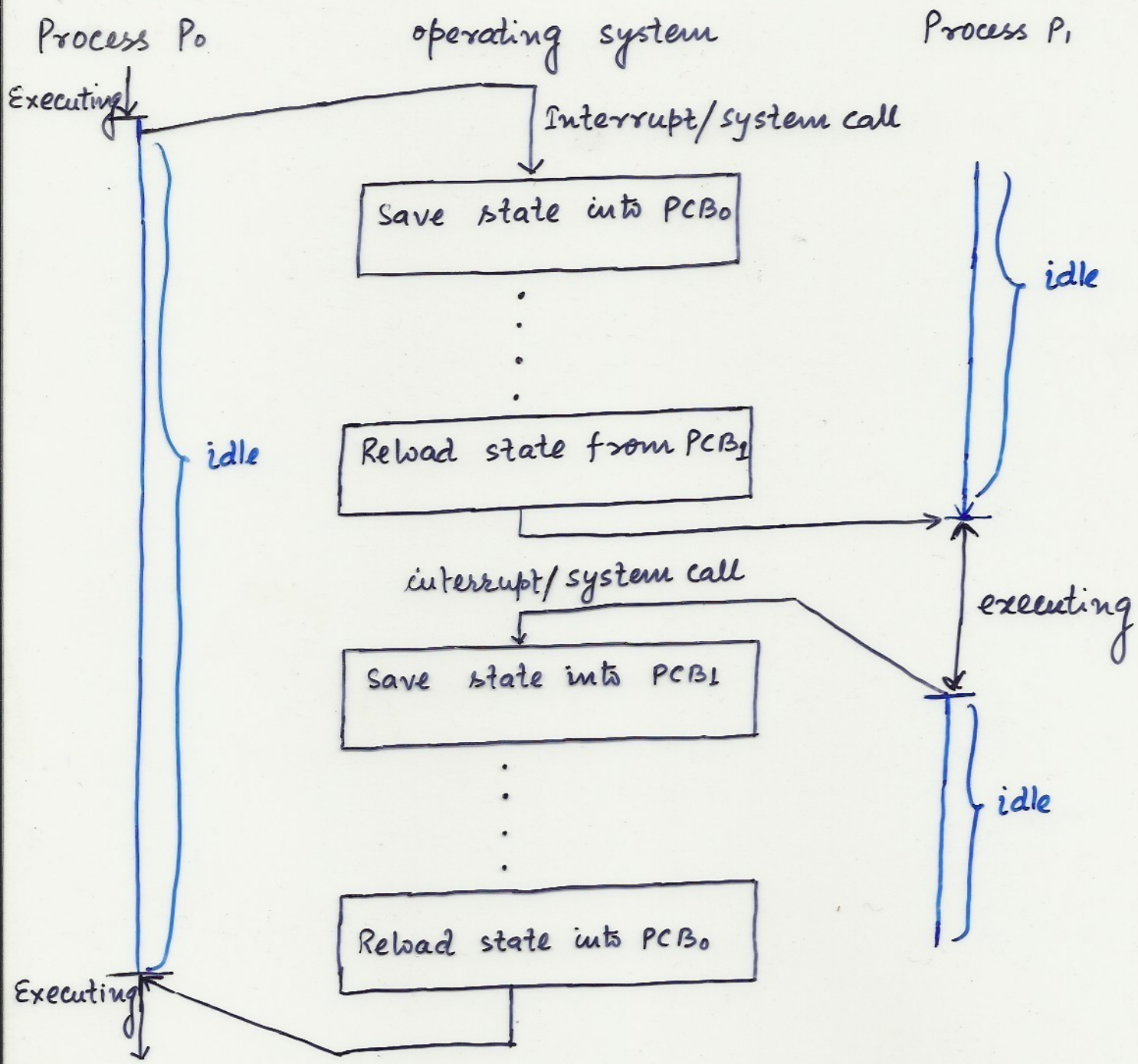
- I/O bound process -
  - * spends more time doing I/O than computations.
  - * Many small CPU bursts.
- CPU bound process -
  - * spends more time doing computations.
  - * few but long CPU bursts.

Context switching

→ When CPU switches to another process, the system must save the state of the old process & load the saved state for the new process.

→ Context-switch time is overhead, - system does no useful work while switching.

→ speed varies from machine to machine, depending on memory speed, no. of registers copied etc.



Context Switching

Process Creation

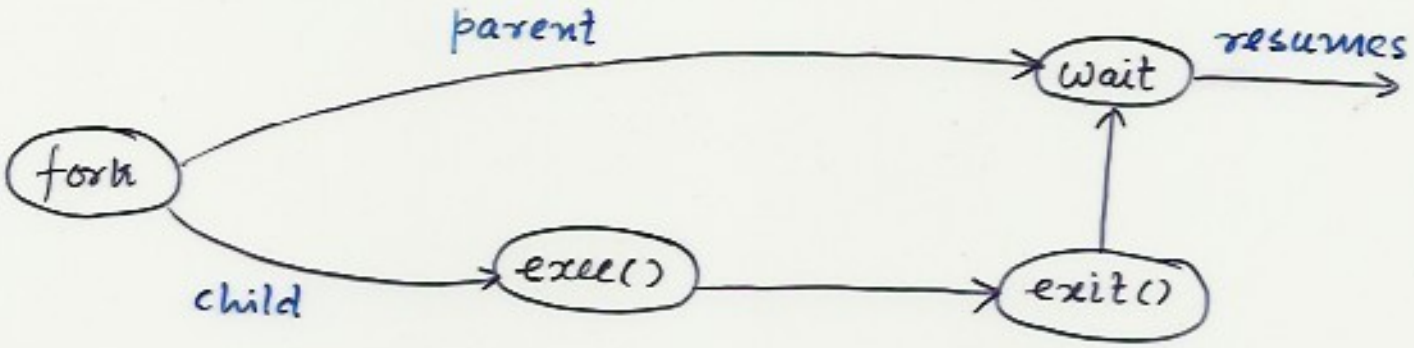
→ Parent process create children processes, which, in turn, create other processes, forming a tree of processes.

→ Resouree sharing
- Parent & children share all resources.
- Children share subset of parent's processes resources.

→ Execution
- Parent & children execute concurrently.
- Parent waits until children terminate.

→ Address space
- child process is a duplicate of parent process.
- child has a program loaded into it.

→ In Unix
- fork system call creates a new process.
- exec system call used after a fork to replace the process memory space with the new process.



Process Termination

→ Process executes last statement and asks the operating system to -

- Output data from child to parent (via wait)
- Process resources are deallocated by operating system.

→ Parent may terminate execution of children process (abort) -

- child has exceeded allocated resources.
- Task assigned to child is no longer required.
- If parent is ~~existing~~ exiting -
  - * Some operating system do not allow child to continue, if its parent terminates
  - * All children terminated - Cascading termination.

Cooperating processes

- Independent processes can not affect or be affected by the execution of another process.
- Cooperating processes can affect or be affected by the execution of another process.
- Advantages of process cooperation :
    * Information sharing
    * Computation speed-up.
    * Modularity.
- Cooperating processes requires mechanism to allow processes to communicate with each other and to synchronize their actions.
- One real example is - Producer-Consumer problem.

Producer- consumer problem

→ Paradigm for cooperating processes.

→ Producer process produces information that is consumed by a consumer process.

→ To run producer & consumer process concurrently, a buffer is required.

- Unbounded buffer places no practical limit on the size of the buffer.
- Bounded buffer assumes that there is a fixed size buffer.

→ Buffer may be provided by the operating system through the use of IPC or explicitly coded by the application programmer using shared memory concept.

Bounded Buffer - Shared Memory solution

- Shared data.

```
#define BUFFER_SIZE 10
typedef struct {
        .....
        .....
        } item;
item buffer[BUFFER_SIZE];
int in = 0;  /* points next free position in buffer */
int out = 0; /* points to 1st full position in buffer */
```

- Maximum BUFFER_SIZE-1, elements can be stored.

Producer Code :

```
while (true)
{ {  ....
     produce an item in item variable.
     ....
     while ((in+1) % BUFFER_SIZE == out)
          ;  /* buffer is full, do nothing */
     buffer[in] = item;
          in = (in+1) % BUFFER_SIZE;
}
```

Consumer process:

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

Interprocess Communication (IPC):

- Unlike shared memory buffer, supported by operating system.
- Mechanism for processes to communicate & to synchronize their actions.
- Best supported by message system.
- IPC facility provides two operations:
  - * send (message)
  - * receive (message)

→ If P & Q wish to communicate, they need to:

- Establish a communication link between them.
- Exchange messages via send/recieve.

→ When a communication link is to be established, some basic points are:

- 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 fixed or variable?
- Is a link unidirectional or bi-directional?

Direct Communication

→ Processes name each other explicitly for comm-unication.

send (P, message) : send a message to process P.

receive (Q, message) : receive " " from " 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.

→ Disadvantage of this scheme is that changing name of a process may necessitate examining all other process definitions.

- All references to the old name must be found so that they can be modified to the new name.

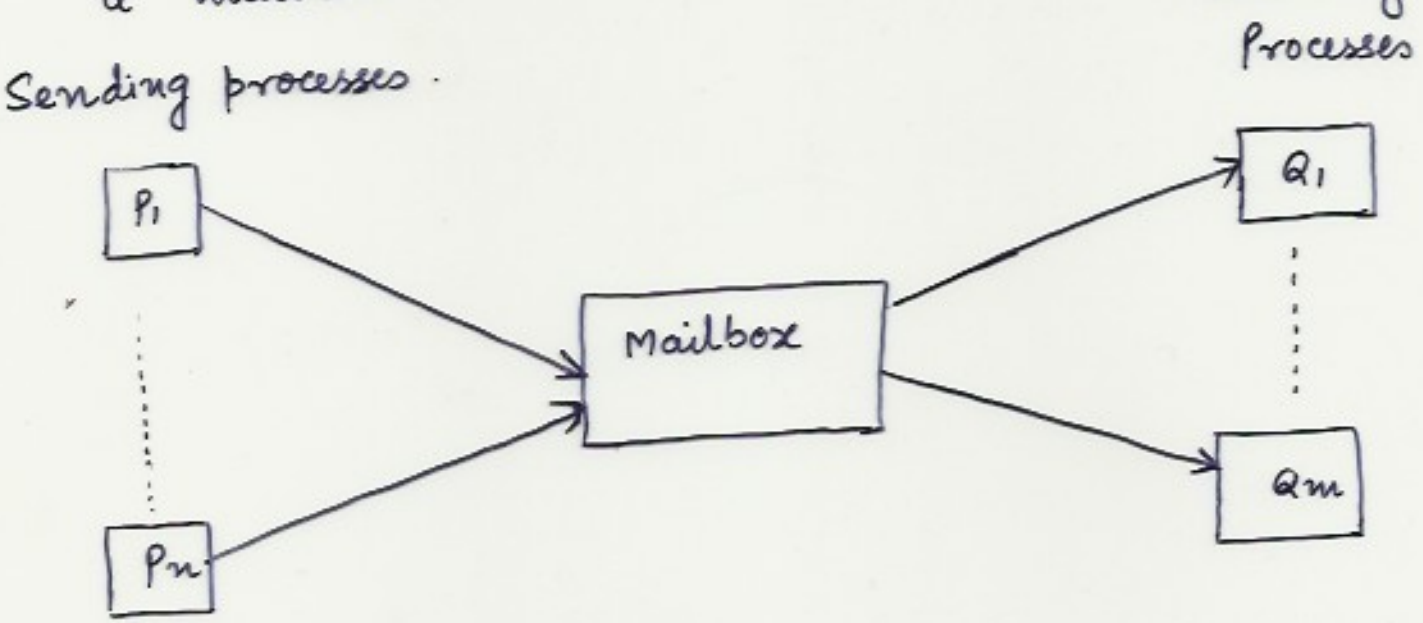
Indirect communication

→ Like direct communication, messages are not sent directly from sender to receiver.

→ Messages are sent to a shared data structure, consisting of queue, called mailbox / port.

→ 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.
- Link may be unidirectional / bi-directional.

→ operations:

- Create a new mailbox.
- Send & 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.

→ Mailboxes may be owned by either the process or the system.

Threads

→ A basic unit of CPU Utilization.

→ Consists of a program counter, a register set and stack.

→ Shares with peer threads its code section, data section & operating system resources.

→ Also called a light weight process.

→ Traditional / heavy weight process is equal to a task with one thread.

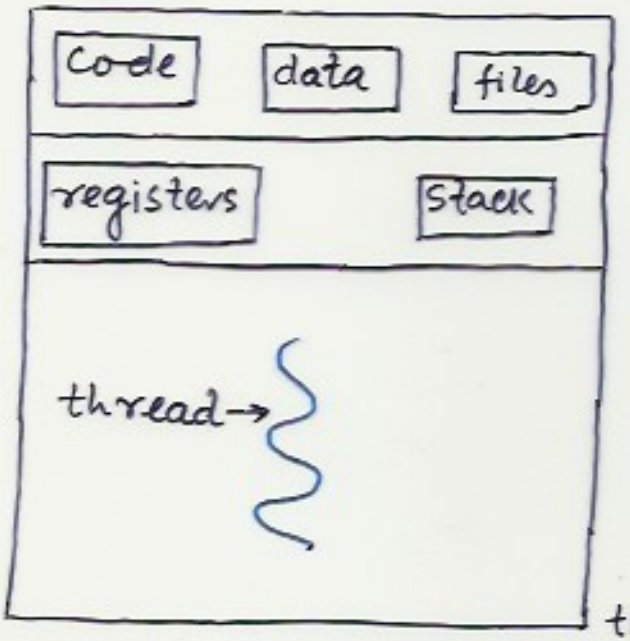
→ A thread must be in exactly one task.

→ Extensive sharing makes CPU switching ~~among~~ among peer threads inexpensive, compared to context switching among heavy weight process-es.

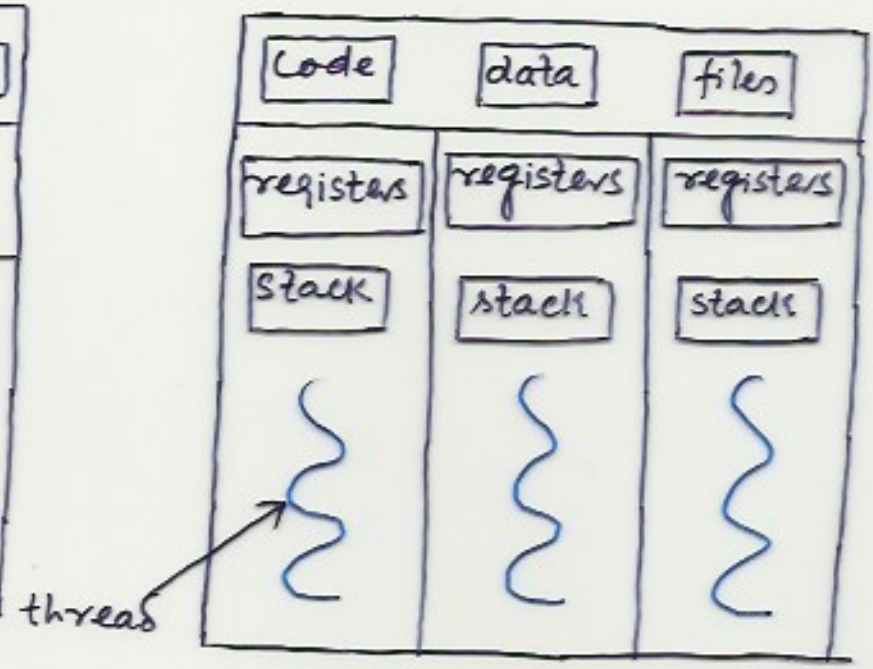
→ Like parallel processing environment -

multithreading a process introduces concurrency control problem, which requires use of critical sections.

→ Like process, a thread can have several states.



Single threaded process.



Multithreaded process.

CPU Scheduling

→ objective of multiprogramming is to maximize CPU utilization.

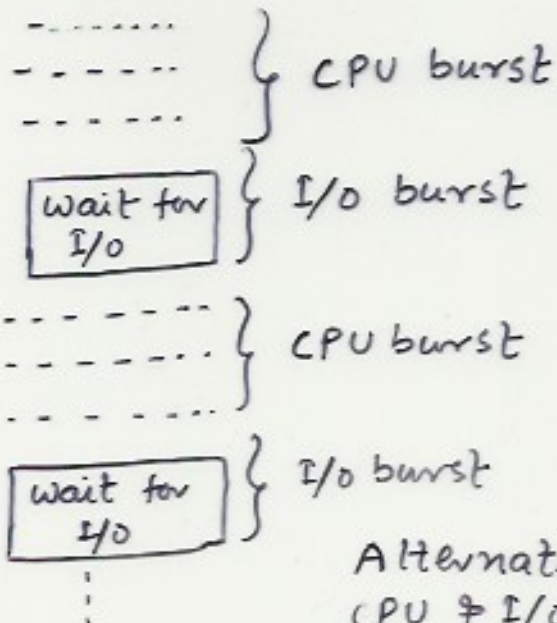
→ If there are more processes, the rest will have to wait until the CPU is free & can be rescheduled.

CPU-I/O Burst cycle:

→ Process execution consists of a cycle of CPU execution & I/O wait.

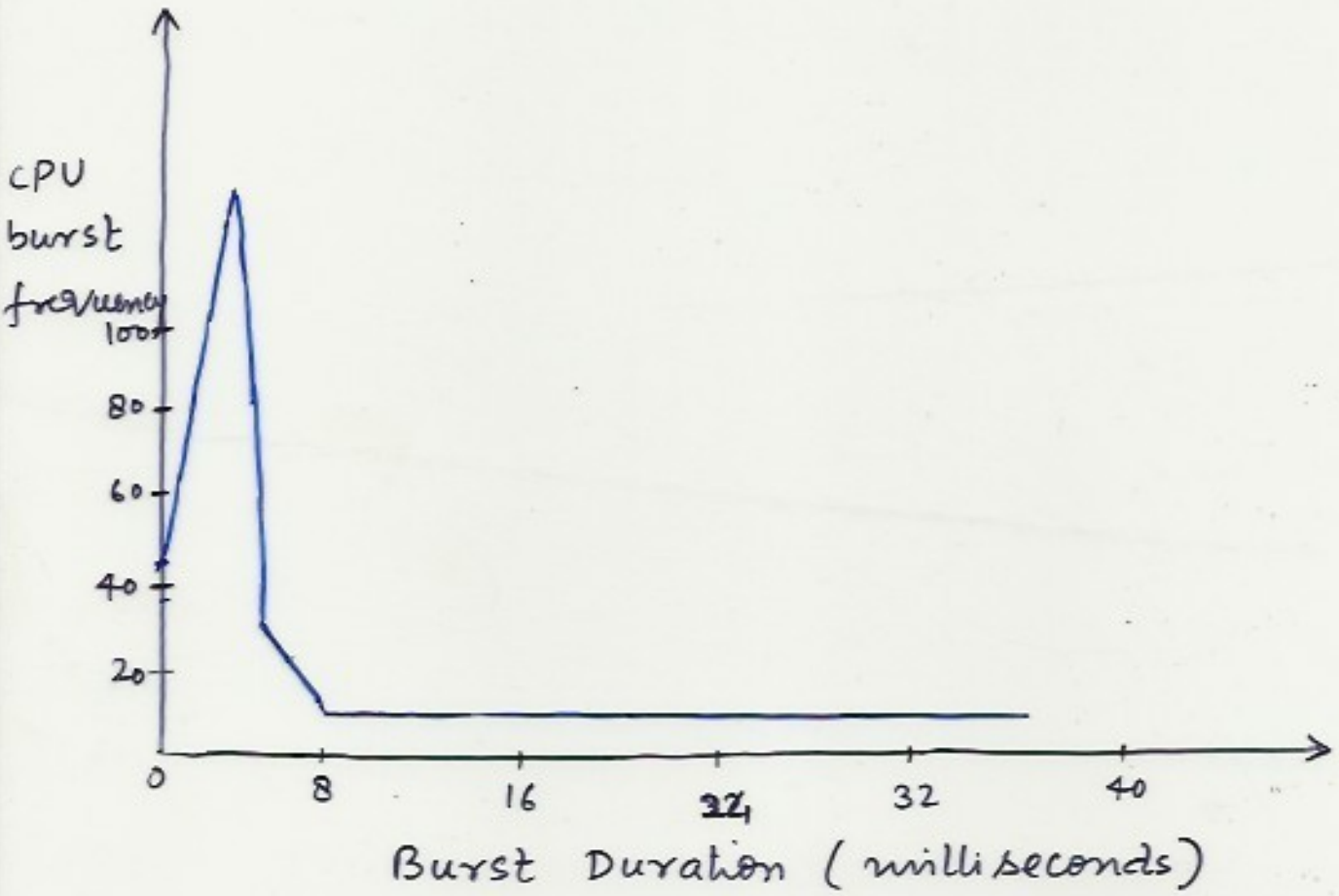
→ Process alternates back & forth between these two states.

→ Process execution begins with CPU burst, followed by I/O burst & so on.



Alternate sequence of CPU & I/O bursts

→ CPU bursts vary greatly from process to process but they tend to have frequency curve similar to that shown below:



→ Large no. of small CPU bursts, & small no. of large CPU bursts.

→ I/O bound program would typically have many short CPU bursts.

→ CPU bound program have a few long CPU bursts.

CPU Scheduler

→ Selects from among the processes in memory that are ready to execute & allocates the CPU to one of them.

→ CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state.
2. Switches from running to ready state.
3. Switches from waiting to ready state.
4. Terminates.

→ Scheduling under 1 & 4 is non-premitive.

→ All other scheduling is premitive.

→ Under non-premitive scheduling -

- Process keeps CPU until it terminates itself or goes to waiting state.

→ Preemptive scheduling incurs extra cost for handling shared data.

Dispatcher :

→ Gives control of the CPU to the process selected by the short-term scheduler.

→ Dispatch Latency -

Time taken by the dispatcher to stop one process & start another.

Scheduling Criteria :

→ Criteria that are used to compare different CPU scheduling algorithms are:

- CPU Utilization -

  Keep the CPU as busy as possible.

- Throughput -

  no. of processes that complete their execution per unit time.

- Turnaround time -

  amount of time to execute a particular process.

- **Waiting Time -**

  amount of time a process has been waiting in the ready queue.

- **Response Time -**

  amount of time it takes from when a request was submitted, until the first response is produced. (For timeshasing systems)

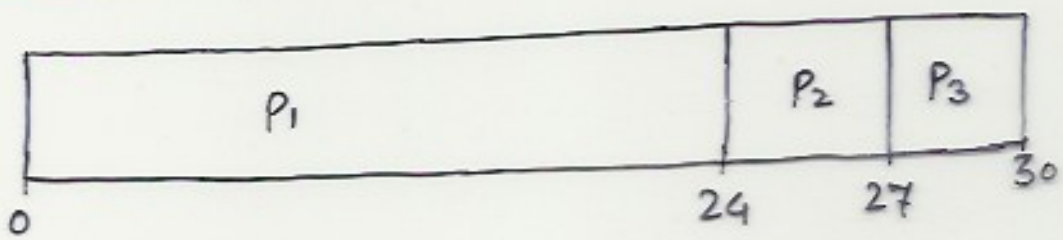
First come First serve (FCFS) Scheduling

→ Process that requests the CPU first, is allocated the CPU first.

Process	CPU burst time
P_1	24
P_2	3
P_3	3

→ Suppose processes arrive in the order P_1, P_2, P_3.

Gantt chart for the schedule is:

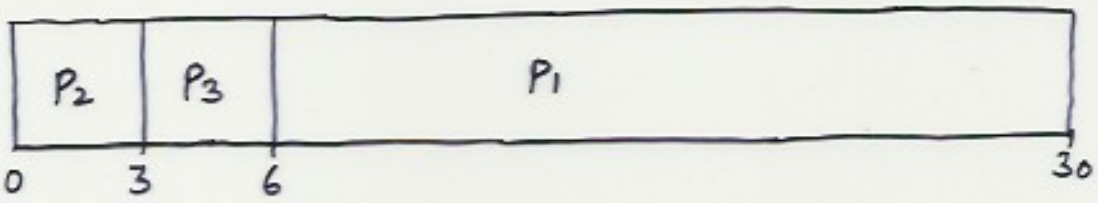


→ Waiting time for $P_1 = 0$, $P_2 = 24$, $P_3 = 27$

→ Avg. waiting time = $(0+24+27)/3 = 17$

→ Suppose processes arrive in the order: P_2, P_3, P_1.

Gantt chart is :



→ Waiting time for $P_1 = 6$, $P_2 = 0$, $P_3 = 3$

→ Avg. waiting time = $(6+0+3)/3 = 3$

→ Much better than previous case.

→ Avg. waiting time for FCFS is not minimal & vary substantially, if processes CPU time vary greatly.

→ FCFS is non-preemptive.

→ Troublesome for time-sharing systems.