Thrashing

→ If a process does not have enough pages, the page fault rate is very high.

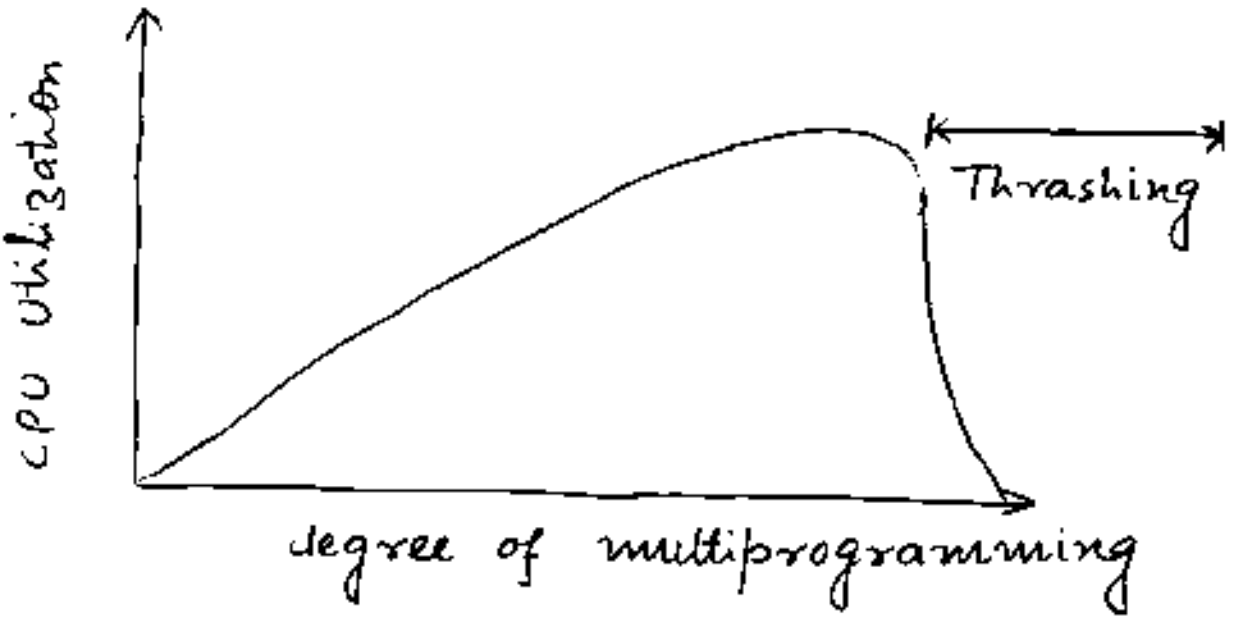
→ This leads to :

- Low CPU Utilization.
- operating system thinks that it needs to increase the degree of multiprogramming.
- So another process is added to the system.

→ Thrasing ≡ A process is busy swapping pages in and out.

→ Global allocation of pages may lead to huge no. of processes causing thrashing.

→ Local allocation limits the no. of processes which are thrashing, but effective access time is increased.



Locality Model :

→ To prevent thrashing :

- concept of locality model is used.

→ As a process executes, it moves from locality to locality.

→ A locality is a set of pages that are actively used together.

→ A program is composed of several different localities, which may overlap.

Working Set Model :

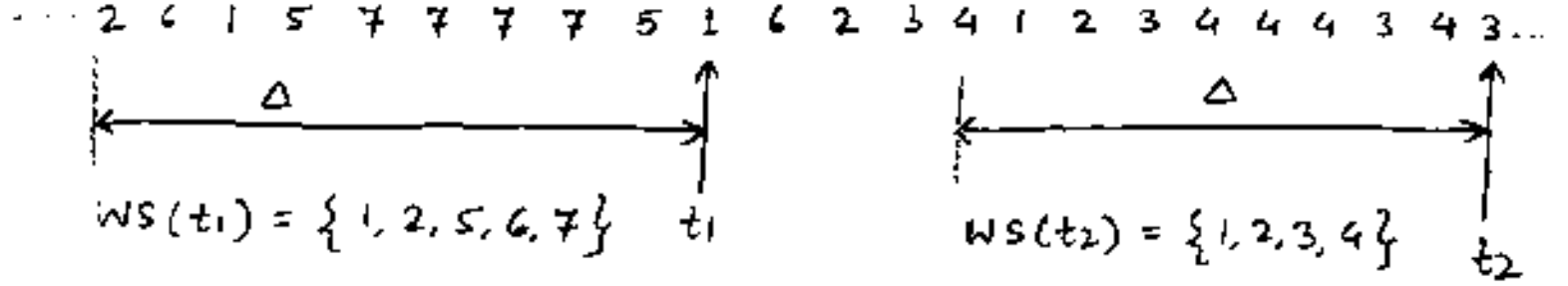
→ Based on the assumption of locality.

→ Uses a parameter $\Delta \equiv$ working set window $\equiv$ a fixed no. of page references.

→ set of pages in the most recent Δ page references is called working set.

→ If a page is currently active, it is in working set else it will be dropped after Δ page references.

page reference table



$\Delta = 10$

working set at time t_1 = {1, 2, 5, 6, 7}

" " " t_2 = {1, 2, 3, 4}

→ Accuracy of the working set depends on the selection of Δ:

- If Δ too small, - it will not cover the entire locality.
- If Δ is too large - it may overlap several localities.
- If Δ is infinit - will encompass entire program.

→ wss_i → working set size of process i.

$$D = \sum_i wss_i$$

where D : total demand of frames

→ If $D > m \Rightarrow$ thrashing, where m : total no. of available frames.

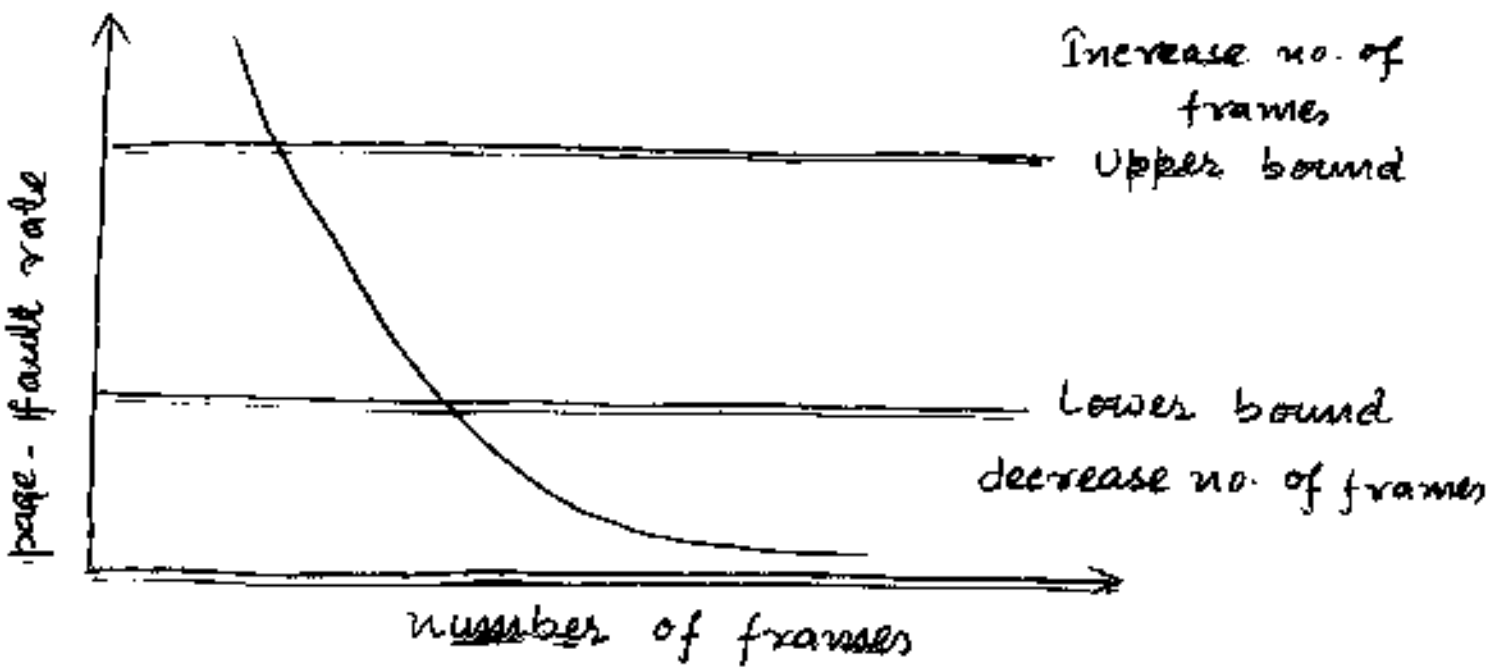
→ If thrashing, suspend one or more processes.

→ Difficulty with working set model is :

- keeping track of the working set.

Page Fault Frevuency

- working set model is successful but clumsy.
- Page Fault Frevuency takes a more direct appro-ach.
- Establishes lower and upper limit of page fault rat.
- If no. of page faults increases & exceeds upper limit a new frame is allocated to that process.
- If page fault rate falls below lower limit, we remove a frame from that process.



Other Considerations:

→ Selection of page replacement algorithm & frame allocation policy are major decisions for demand paging.

→ Some other considerations are there:

- Prepaging.
- Page size.
- ~~Inverted page table.~~
- Program structure.

Prepaging:

→ Reduces the large no. of page faults that occurs at process startup.

→ Prepage all or some of the pages a process will need, before they are referenced.

→ Advantageous in some cases.

→ Question is -

Whether the cost of prepaging is less than servicing the corresponding page faults?

→ Assume that

s: no. of pages that are prepaged.

α ($0 \leq \alpha \leq 1$): a fraction of these pages, which are actually used.

Whether the cost of α saved page faults is greate than the cost of prepaging $(1-\alpha)$ unnecessary pages

→ If α is close to 0 ⇒ prepaging loses.

Page Size :

→ Selection of page size must take into consideration

- Page table size
  - * larger the page, smaller page table.
- Fragmentation
  - * Larger the page, larger the internal fragmentation.
- I/O overhead
  - * Larger the page size, smaller is the I/O overhead time.

- Locality

  * A smaller page allows each page to match program locality more accurately.

Program structure :

→ Demand paging is transparent to the user.

→ System performance can be evolved, if an awareness of the underlying demand paging is done.

```
for ( j = 0;  j < 128;  j++)
        for ( i = 0;  i < 128;  i++)
                a[i][j] = 0;
```

Column-major order initialization

Variation 1

```
for ( i = 0;  i < 128;  i++)
        for ( j = 0;  j < 128;  j++)
                a[i][j] = 0;
```

Row-major order initialization.

Variation 2

assume page size of 128 words.

Variation 1 will require nearly $128 \times 128 = 16,384$ page faults, if allocated less than 128 frames

Variation 2 will require only 128 page faults.

→ Careful selection of program structures can increase locality & hence lower the page fault rate.

Process Management

- A process can be thought of as a program in execution.
- A process needs certain resources - CPU time, memory, files & I/O devices - to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.
- operating system is responsible for following activities in process management:
  - * creation & deletion of both user & system processes.
  - * scheduling of processes.
  - * provision of mechanisms of synchronisation
  - * communication & deadlock handling for processes.

Process:

- Informally, a program in execution.
- A process is more than a program code:
- It includes the current activity in the form of program counter & contents of the processor registers.
- It also includes stack containing temporary data.

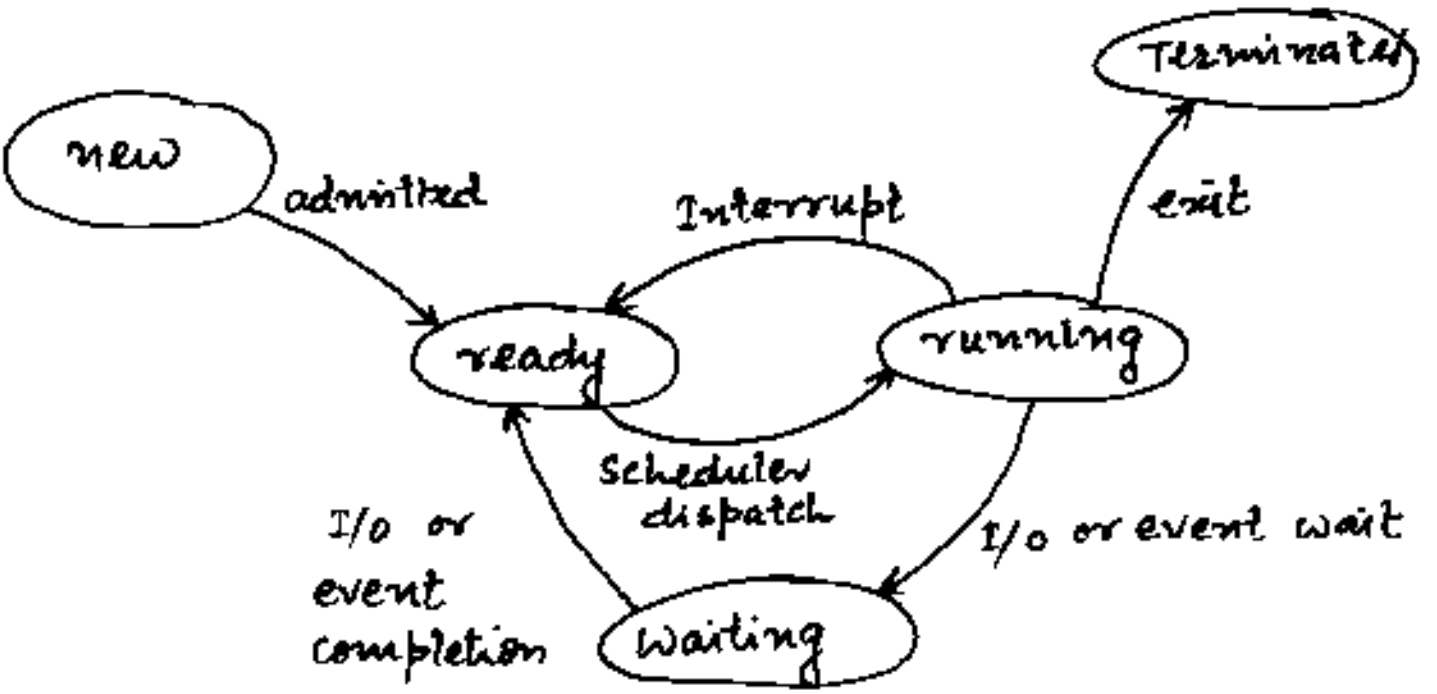
Process States

- State is defined in part by the current activity of that process.
- As process executes - it changes its state
- States of a process can be:
  - * New - process is being created.
  - * Running - Instructions are being executed

* Waiting - waiting for some event to occur.

* Ready - waiting to be assigned to CPU.

* Terminated - finished execution.

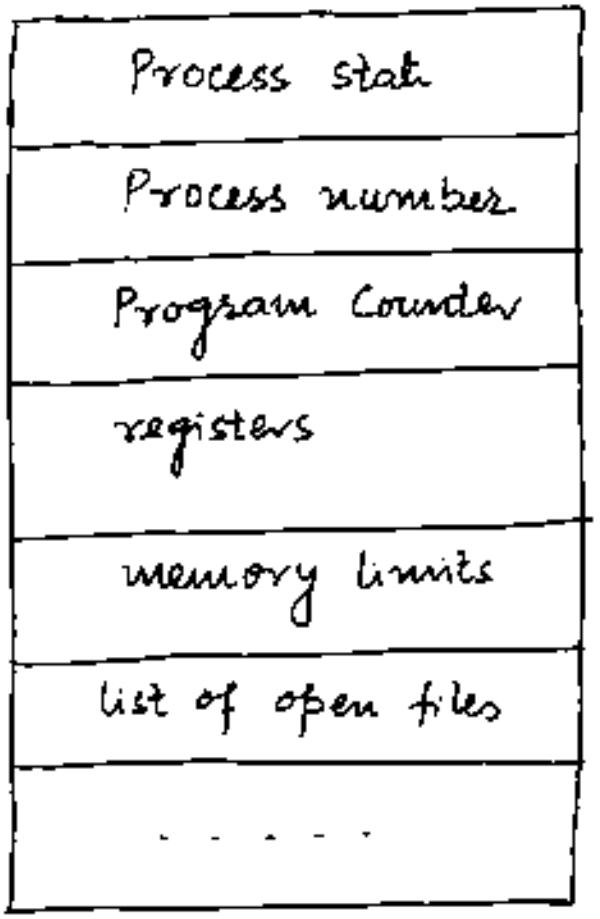


Process State Diagram

Process Control Block (PCB)

→ Information associated with each process:

- Process state
- Program Counter
- CPU Registers
- CPU scheduling information.
- Memory management Information.
- Accounting information.
- I/O status information.



Process Scheduling Queues:

job Queue:

→ As process enter the system, they are put into job Queue.

→ consists of all processes in the system.

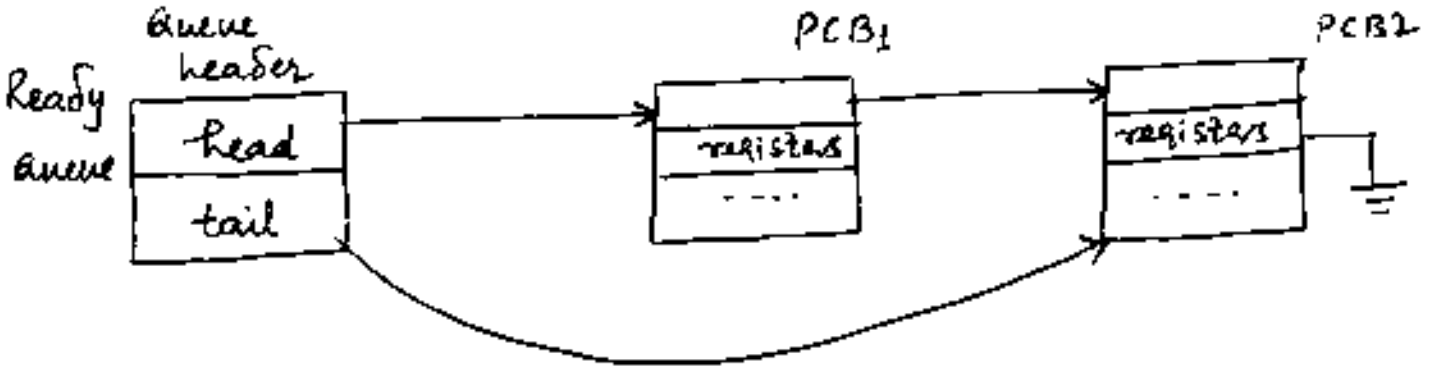
Ready Queue:

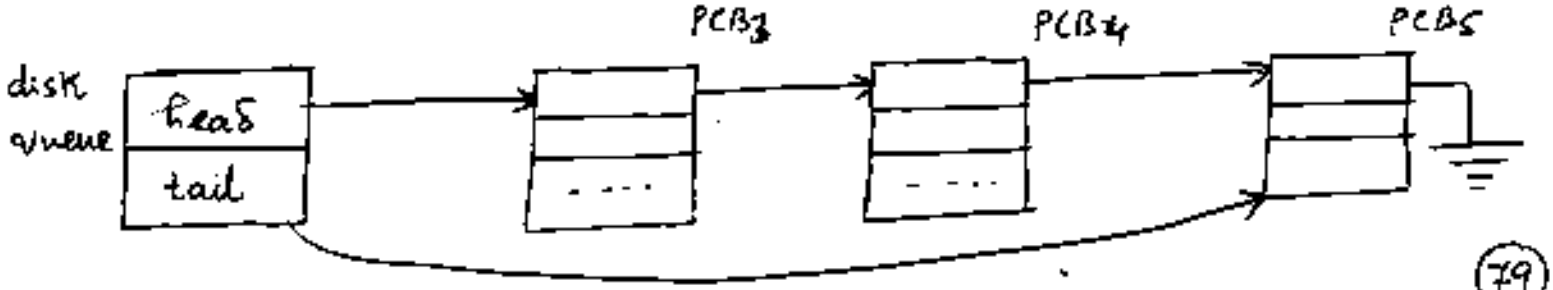
→ set of all processes residing in main memory, ready and waiting to execute.

Device Queue:

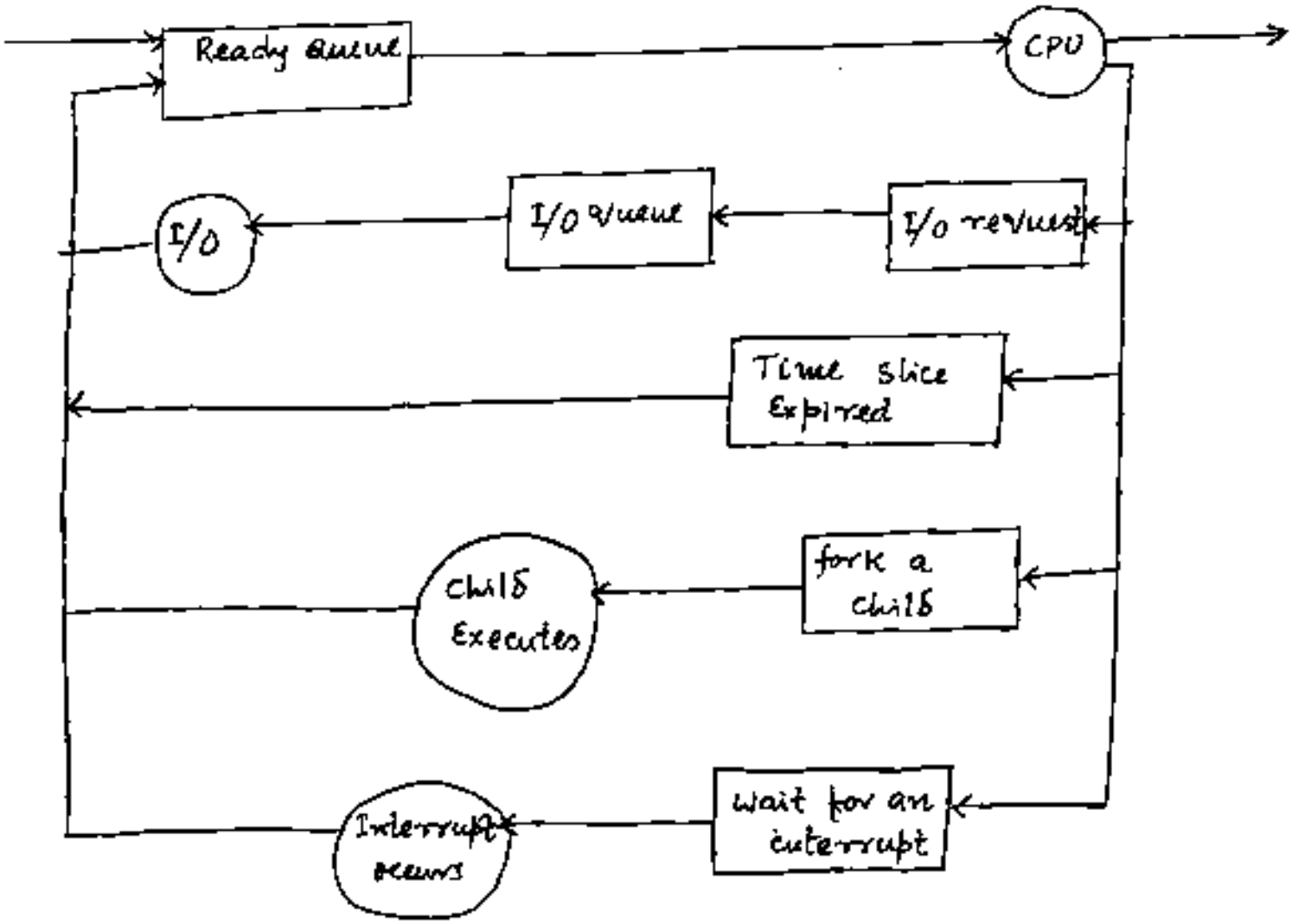
→ set of all processes waiting for an I/O device.

→ Each device has its own Queue.





Queueing - diagram representation of process scheduling:



→ Once the process is allocated the CPU, & is executing, one of the several events could occur:

1.) Process could issue an I/O request & placed in an I/O queue.

2.) Process could create a new subprocess & wait for its termination.

3.) Process could be removed from the CPU due to an interrupt & be put in ready queue.

Schedulers

→ A process migrates between various scheduling queues throughout its life time.

→ Operating system must select processes from these queues.

→ Selection is carried out by the appropriate scheduler.

1. Long Term Scheduler:

- Also called job scheduler.
- Selects which process should be brought into ready queue.
- Invoked very infrequently
- Controls degree of multiprogramming.

2. Short Term Scheduler:

- Also called CPU scheduler.
- Selects which process should be executed next & allocates CPU.
- Invoked very frequently.