Waiting time for $P_4 = (5 - 3) = 2$

Average waiting time $= \frac{(9 + 0 + 15 + 2)}{4} = 6.5$ ms

### (d) Priority Scheduling

**Advantages**

- A priority is associated with each process and CPU is granted to the process of highest priority. Equal priority processes are scheduled in FCFS manner.
- Priorities may be defined on the basis of number of open files, ratio of average I/O burst to average CPU burst time memory requirements etc.
- Priority scheduling can be pre-emptive or non-pre-emptive.

**Disadvantage**

- Starvation may result. A low priority process may have to wait indefinitely for CPU. (Aging is used to solve this problems; the priorities of waiting process is gradually increased).

### (e) Round Robin Scheduling

- A time quantum (10 to 100 ms) is selected. Each process in the ready queue is allotted the CPU for this time quantum. The ready queue is treated as a circular queue.
- The process may release the CPU, when it performs I/O or when it finishes executing before the time quantum has expired. Then, next process is selected by the scheduler from the ready queue allotted to the CPU.

### (f) Multilevel Queue Scheduling

- The ready queue is partitioned into separate queues processes are assigned a queue according to a priority assigned to the process.
- Foreground processes are assigned a higher priority and the background processes are assigned lower priority.

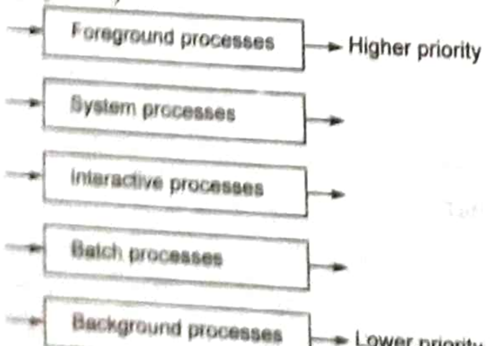


Fig. 3

### (g) Multilevel feedback queue scheduling

- This is similar to multilevel queue scheduling except that the processes are allowed to move between queues. If a process was too much CPU time, it is moved to a low priority queue. Also a process which has waited a long time in a low priority queue is moved up to a high priority queue.

## Process Synchronization and Concurrency

Concurrent systems can run a collection of processes concurrently. Concurrent processing is the basis of multiprogrammed operating systems. The system must provide for mechanisms of process synchronization and communication to support concurrent execution of processes.

### Critical Section

- A critical section is a segment of code during which a process might be changing common variables, updating a table etc. The execution of critical sections by processes should be mutually exclusive in time.
- The problem is to determine a mechanism to allow various processes to execute their critical sections. The solution should satisfy the following conditions :

#### (a) Mutual Exclusion

If a process $P$ is executing its critical section then, no other process should be executing in their critical section.

#### (b) Progress

Consider a case when no process is executing in its critical section and some processes exist which wish to execute their critical sections. In such a case, the decision that which process is allowed to enter the critical section is not made by processes which are executing their remainder sections.

#### (c) Bounded Waiting

There must exist a bound on the number of times that the other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before the request is granted.

### The Bakery Algorithm

This gives a solution to the critical section problem with multiple processes.

```
Repeat
  Choosing [i] = true,
  Number [i] = max. (number [0], number[1]
  ...number [n − 1]) + 1;
  Choosing [i] = false;
  for j = 0 to n − 1
  do begin
    while choosing [i] do no-op;
    while number [j] ≠ 0 and (number [j], j) <
    (number [i], i) do no-op;
  end;
  critical section
  number [i] = 0;
  remainder section;
until false
```

**Explanation** On entering the system each process receivers a number. The process with the lowest number is allowed to enter the critical section first in case two processes have the same number the processes with lowest process-id is served first.

## Semaphores

- A semaphore is a integer variable that apart from initialisation can be accessed only through two standard atomic operations wait and signal.

  Wait (s) : while $S \le 0$ do no-op

  $$S = S - 1$$

  Signal(s) : $S = S + 1$
- Wait and signal operations are executed individually.
- Semaphores can be used to solve various synchronization problems. The no-op in wait causes busy waiting. To prevent that a process is made to wait while it finds the value of the semaphore $S \le 0$.

  The blocked process is put in a waiting queue associated with the semaphore. Then, the signal operation wakes up all the process waiting on the semaphore queue.

## Interprocess Communication

Two schemes are popular for interprocess communication–

1. Shared memory
2. Message system

### 1. Shared Memory

Shared memory systems require communicating process to share their variables. The process share information by the use of shared variables. The responsibility of providing communication rests with the application programmers. The operating system only provides the ways to access and create shared memory.

### 2. Message System

The message systems allow processes to exchange messages. The responsibility of providing communication rests with the operating system. Two operations are provided send (message), receive (message). Messages send by processes can be fixed size or variable size.

## Deadlocks

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources, if the resources are not available at that time, the process enters in a wait state.
- It may happen that waiting process will never change state because the resources it have requested, are held by other waiting processes. This situation is called deadlock.
- A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its designated task.
- The number of resources requested may not exceed the total number of resources available in the system *i.e.,* a process cannot request three printers, if the system has only two printers. Under the normal mode of operation, a process may utilize a resource in only the following sequence :

  (i) **Request** If the request cannot be granted immediately then, the requesting process must wait until it can acquire the resource.

  (ii) **Use** The process can operate on the resource.

  (iii) **Release** The process releases the resource.

## Necessary Conditions for Deadlock

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

### 1. Mutual Exclusion Condition

Resources must be allocated to processes at any time in an exclusive manner and not on a shared basis for a deadlock to be possible. If another process requests that resource, the requesting process must be delayed until the resource has been released.

### 2. Hold and Wait Condition

Even if a process holds certain resources at any moment, it should be possible for it to request for new ones. It should not have to give up the already held resources to be able to request for new ones. If it is not true, a deadlock can never take place.

### 3. No Pre-emption Condition

Resources cannot be pre-empted it, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

### 4. Circular Wait Condition

- There must exist a set or $\{P_0, P_1, P_2, \ldots, P_n\}$ waiting processes such that $P_0$ is waiting for a resource that is needed by $P_1$, $P_1$ is waiting for a resource that is held by $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by $P_n$ and $P_n$ is waiting for a resource that is held by $P_0$.
- It is necessary that all these four conditions have to be satisfied simultaneously for the existence of a deadlock. If one of them does not exist, deadlock can be avoided.

## Resource Allocation Graph

- The resource allocation graph consists of a set of vertices $V$ and a set of edges $E$.
- The set of vertices $V$ is partitioned into two different type of nodes $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the active processes in the system and $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system. $A$ directed edge from process $P_i$ to resource type $R_j$, is denoted by $P_i \rightarrow R_j$, it signifies that process $P_i$ requested an instance of resource type $R_j$ and in currently waiting for that resource.
- A directed edge from resource type $R_j$ to process $P_i$ is denoted by $R_j \rightarrow P_i$, it signifies that an instance of resource type $R_j$ has been allocated to process $P_i$.
- A directed edge $P_i \rightarrow R_j$ is called a request edge, a directed edge $R_j \rightarrow P_i$ is called an assignment edge.

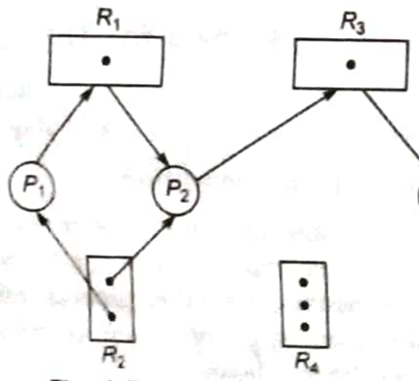


**Fig. 4** Resources allocation graph

### Resource Instance

1. One instance of resource type $R_1$.
2. Two instances of resource type $R_2$.
3. One instance of resource type $R_3$.
4. Three instances of resource type $R_4$.

### Process States

1. Process $P_1$ is holding an instance of resource type $R_2$ and is waiting for an instance of resource type $R_1$.
2. Process $P_2$ is holding an instance of $R_1$ and $R_2$ and is waiting for an instance of resource type $R_3$.
3. Process $P_3$ is holding an instance of $R_3$.

- If the graph contains no cycles then no process in the system is deadlock and if the graph contain the cycle then a deadlock may exist.

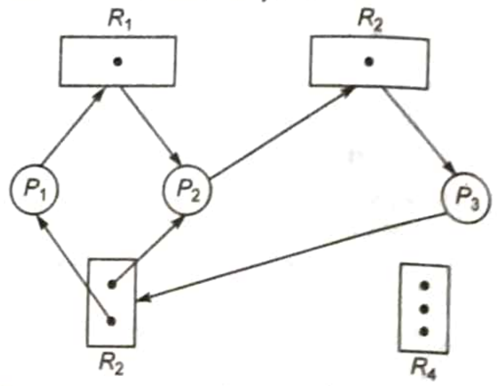


**Fig. 5** Resource allocation graph with a deadlock

- If each resource type has several instances then a cycle does not necessarily imply that a deadlock occur. In this case, a cycle in the graph is a necessary but not sufficient condition for the existence of deadlock.

## Deadlock Handling Strategies

Various strategies have been followed by different operating system to deal with the problem of deadlock, which ignore a deadlock, detect a deadlock recover from deadlock, prevent and avoiding a deadlock.

### 1. Deadlock Prevention

Deadlock prevention is a set of methods for ensuring that atleast one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how request for resources can be made.

### 2. Deadlock Avoidance

A deadlock avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular wait condition can never exist. The resource-allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

**e state**

A state is safe, if the system can allocate resources to each process and still avoid a deadlock.

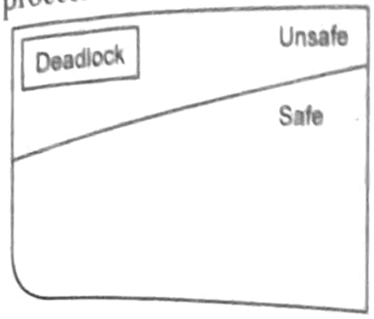


Fig. 6 Safe, unsafe and deadlock

- A system is in a safe state only if there exists a safe sequence. A safe state is not a deadlock state.
- A deadlock state is an unsafe state. Not all unsafe stages are deadlocks.

**nker's algorithm**

- **Date structures for the banker's algorithm**

  **Available** Vector of length $m$. If available $[j] = k$, there are $k$ instances of resource type $R_j$ available.

  **Max** $n \times m$ matrix. If max $[i, j] = k$, then process $P_i$ may request at most $k$ instances of resource type $R_j$.

  **Allocation** $n \times m$ matrix. If allocation $[i, j] = k$, then $P_i$ is currently allocated $k$ instances of $R_j$.

  **Need** $n \times m$ matrix. If need $[i, j] = k$, then $P_i$ may need $k$ more instances of $R_j$ to complete its task.

  Need $[i, j]$ = max $[i, j]$ – allocation $[i, j]$

- **Safety algorithm**

  (i) Let work and finish be vectors of length $m$ and $n$, respectively. Initialize–

  Work = Available

  Finish $[i]$ = False (for $i = 1, 2, \ldots, n$)

  (ii) Find $i$ such that both–

  (a) Finish $[i]$ = False

  (b) Need $\leq$ Work

  If no such $i$ exists, go to step 4.

  (iii) Work = Work + Allocation

  Finish $[i]$ = Tree

  Go to step 2.

  (iv) Finish $[i]$ = True (for all $i$)

  Then, the system is in a safe system.

**Deadlock Detection**

- Allow system to enter deadlock state and then there is–
  → Detection algorithm
  → Recovery scheme
- Single instance of each resource type
  → Maintain wait for graph–

  Nodes are processes

  $P_i \to P_j$ if $P_i$ is waiting for $P_j$.

  → Periodically invoke an algorithm that searches for a cycle in the graph.

  → An algorithm to detect a cycle in a graph requires an order of $n^2$ operations, where $n$ is the number of vertices in the graph.

**Resource allocation graph and wait of graph**

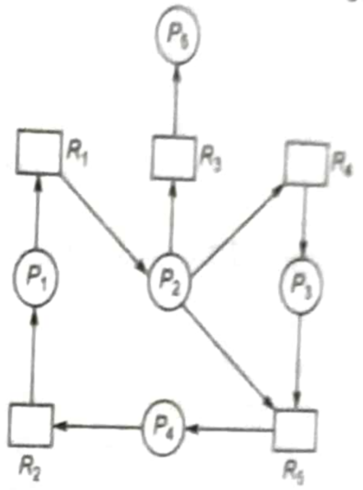


Fig. 7 Resource allocation graph

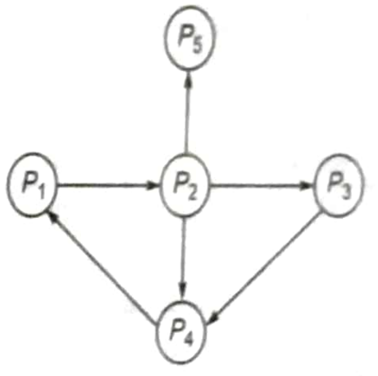


Fig. 8 Corresponding wait for graph

**Recovery scheme**

(i) Resource pre-emption

(ii) Process terminations

→ Abort all deadlock processes.

→ Abort one process at a time until the deadlock cycle eliminated.

## Memory Management

Memory management techniques allow several processes to share memory. When severed processes are in memory they can share the CPU, thus increasing CPU utilization.

### Overlays

This techniques allows to keep in memory only those instructions and data which are required at given time. The other instruction and data is loaded into the memory space occupied by the previous ones when they are needed.

## Swapping

Consider an environment which supports multiprogramming using say Round Robin CPU scheduling algorithm. Then, when one process has finished executing for one time quantum, it is swapped out of memory to a backing store. The memory manager then picks up another process from the backing store and loads it into the memory occupied by the previous process. Then, the scheduler picks up another process and allocates the CPU to it.

## Logical versus Physical Address Space

An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit is commonly referred to as a physical address.

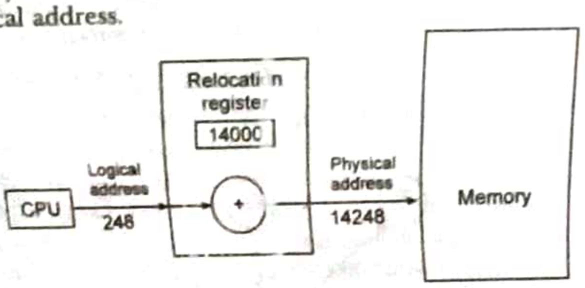


Fig. 9

## Dynamic Loading

- As we know that the entire program and data of a process must be in physical memory for the process to execute. The size of a process is limited to the size of physical memory.
- To obtain better memory-space utilization, we can use dynamic loading.
- With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then, control is passed to the newly loaded routine.

## Memory Management Techniques

1. **Single Partition Allocation**
   - The memory is divided into two parts. One to uses the operating system and the other is for user programs.
   - The operating system code and date is protected from being modified by user programs using a base register.

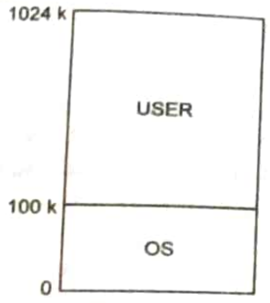


Fig. 10

2. **Multiple Partition Allocation**

**(i) Fixed partition scheme**

- Memory is divided into a number of fixed size partitions. Then, each partition holds one process.
- This scheme supports multiprogramming as a number of processes may now be brought into memory and the CPU can be switched from one process to another.
- The degree of multi-programming is however limited by the number of partitions. The size and number of processes in memory is limited by the size and number of partitions.
- When a process arrives for execution, it is put into the input queue of the smallest partition which is large enough to hold it.

**(ii) Variable partition scheme**

- A block of available memory is designated as a hole at any time, a set of holes exists which consists of holes of various sizes scattered throughout the memory. When a process arrives and needs memory, this set of holes is searched for a hole which is large enough to hold the process. If the hole is too large, it is split into two parts, the unused part is added to the set of holes. All holes which are adjacent to each other are merged.
- There are several algorithms which are used to search for the hole in the set.

  (a) **First-fit** This allocates the first hole in the set which is big enough to hold the process.

(b) **Next-fit** This works like the first-fit algorithms except that it keeps track of the position of the hole found in the set. The next time it is called it starts searching from that position.

(c) **Best-fit** This allocates the smallest hole which is large enough to hold the process.

(d) **Worst-fit** This algorithm simply allocates the largest hole.

### Disadvantages

The above schemes cause external and internal fragmentations of the memory.

#### (i) External fragmentation

It is said to exist when there is enough total memory in the system to satisfy the requirements of a process but the memory is not contiguous.

#### (ii) Internal fragmentation

The memory wasted inside the allocated blocks of memory called internal fragmentation.

*e.g.*, Consider a process requiring 150 k, thus if a hole of size 170 k is allocated to it the remaining 20 k is wasted.

**Compaction** This is a strategy to solve the problem of external fragmentation. All free memory is placed together by moving the processes to new locations.

## Paging

- Paging is a memory management technique which allows the memory.
- This allows the memory to be allocated to the process wherever it is available.
- Physical memory is divided into fixed size blocks called frames. Logical memory is broken into blocks of same size called pages. The backing store is also divided into same size blocks.

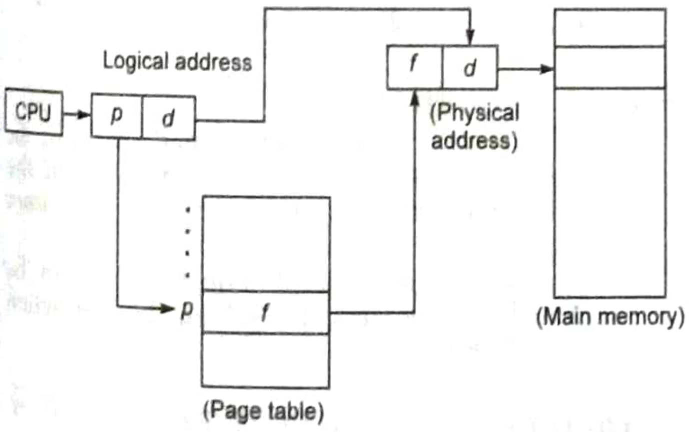


**Fig. 11**

- When a process is to be executed its pages are loaded into available page frames. A frame is a collection of contiguous pages. Every logical address generated by the CPU is divided into two parts.
- The page number ($p$) and the page offset ($d$). The page number is used as an index into a page table. Each entry in the page table contains the base address of the page in physical memory ($f$). The base address from $p$th entry is then combined with the offset ($d$) to give the actual address in memory.

## Implementation of the Page Table

- Page tables are usually implemented as a set of associated registers [or Translation Look Aside Buffers (TLAB)]. This set of registers is made up of very high speed memory but is expensive.
- Each associative register consists of a key value and a field. An input item is compared with key value in all the registers in the set simultaneously. If the key value in any register matches the corresponding field in the register is output.
- The key value in the associative register is the page number. The item is the base address of the page in physical memory.
- The associate registers contain only a few of the page table enteries. (The page table is stored in its entirely in physical memory.)
- The page number generated by the CPU is presented to the associative memory. If the page is found (this is known as hit), the frame number is immediately made available. If, however, the page number is not found in the associative memory (then it is called a miss) a memory reference to the page table is made and frame number is obtained. This page number and frame number are also added to the associative memory.

### Advantages of Paging

(i) Allows the memory of a process to be non-contiguous. This also solves the problem of fitting or storing memory blocks of varying sizes in secondary memory (such as backing store).

(ii) Paging allows sharing of common code. Two or more processes are allowed to execute the same code at the same time.

## Virtual Memory

- Separation of user logical memory from physical memory. It is technique to run process size more than main memory.

- Virtual memory is a memory management scheme which allows the execution of a partially loaded process.
- The term virtual memory is normally associated with the ability to address a storage space much larger than that available in the primary storage of a computer system.

### Advantages

(i) A very large virtual address space is made available to a program, so the size of a process is no longer limited by the size of the physical memory.

(ii) Since, each user now needs less physical memory to run, more user can be now run at the same time.

(iii) Less I/O is required to load or swap a process in memory so each user can run faster.

## Demand Paging

- Virtual memory technique is usually implemented using demand paging.
- Demand paging is a combination of swapping and paging.
- When a program is to be executed, it is brought into main memory page by page *i.e.,* the swapper (called the lazy swapper) brings in a few pages from the backing store and the program starts execution. Entire program lies on the backing store.

Effective access time for demand paging

$$= (1 - P) \times ma + P \times \text{Page fault time}$$

where, $P$ is the probability of a page fault and *ma* is memory access time.

## Page Replacement

In a multiprogramming environment, the following scenario often results :

→ While execution of a process, a page fault occurs and there are no free frames on the free frame list. This is called over allocation of memory and results due to increase in the degree of multi programming.

→ Page replacement is a technique to solve this problem.

### Concept

- If no free frame is available, a frame is found which is not in use. The contents of the frame are written on to a backing store and the page tables are modified to indicate that the page is no longer in memory. Thus, the frame can be used to hold the page for which the page fault was generated.
- One bit is associated with each frame. If the bit is 1, this implies that the contents of the page was modified. This is called the dirty bit. If the dirty bit is 0, the content of the frame need not be written to the backing store.

## Page Replacement Algorithms

1. **FIFO** (First In First Out)
   - Each page in the memory is associated with a time when it was brought into memory. The oldest page is chosen. A queue is maintained for the pages. When a page is brought into memory, it is inserted at the tail of the queue. A page is replaced at the head of the queue.
   - The FIFO algorithm is simple to implement. The performance though is not very good. The algorithm suffers from Belady's anamoly means that the page fault rate may increase as the number of frames allocated increases.

2. **Optimal Page Replacement**
   - The page which will not be used for a longest period of time is replaced.
   - This algorithm gives a lowest page fault rate. It is very difficult to implement because it requires future knowledge about the usage of the page.

2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2

$\vdash\!\!\longrightarrow\!\!\dashv\; t_1 \qquad \vdash\!\!\longrightarrow\!\!\dashv\; t_2$

Working set at $t_1$ = {1, 2, 5, 6, 7}

Working set at $t_2$ = {3, 4}

3. **Least Recently Used** (LRU)
   - The page which has not been used for a longest period of time is replaced.
   - LRU algorithm needs considerable hardware assistance for implementation of this strategy.

## Frame Allocation

- The maximum number of frames that can be allocated to a process depend on the size of the physical memory (*i.e.,* total number of frames available).
- The minimum number of frames which can be allocated to a process depend on the instruction set architecture.

## Thrashing

- A process is said to be thrashing when it is spending more time in paging (*i.e.,* it is generating a lot of page faults) than executing.

- Thrashing causes low CPU utilization. The processes which are thrashing queue up for the paging device.

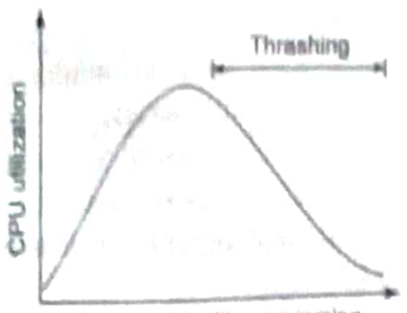


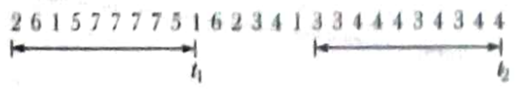
Fig. 12

- The CPU schedule sees the empty ready queue and tries to increase the degree of multiprogramming by introducing more processes in the system. These new processes cause more page faults and increase the length of the queue for the paging device.

## Working Set Model

- The working set strategy is used to prevent thrashing. It is based on the assumption of locality.
- The locality model states that as a process executes it moves from one locality to other.
- A locality is a set of pages that are activity used together. A program is composed of several localities which may overlap.
- If a process is allocated enough frames to satisfy the current locality, then once all the pages in the current locality are in the memory, the process will not page fault till it moves to another locality.
- The working set model uses a parameter Δ to define a working set window. A working set is comprised of all those pages which are referenced in the most recent Δ page references.
- A page which is not in active use will be removed from the working set Δ time units after its last reference.

*e.g.*, For $\Delta = 10$ consider the list of page references



Working set at $t_1 = \{1, 2, 5, 6, 7\}$
Working set at $t_2 = \{3, 4\}$

## File Management

- Logically related data items on the secondary storage are usually organized into named collections called files.
- A file many contain a report, an executable program, as a set of commands to the operating system.
- A file often appears to the users as a linear array of characters or record structures.
- The file system consists of two parts
  (i) A collection of files
  (ii) A directory structure
- The file management system can be implemented as one or more layers of the operating system.
- The common responsibilities of the file management system includes the following :
  1. Mapping of access requests from logical to physical file address space.
  2. Transmission of file elements between main and secondary storage.
  3. Management of the secondary storage, such as keeping track of the status, allocation and deallocation of space.
  4. Support for protection and sharping of files and the recovery and possible restoration of files after system crashes.

## File Attributes

Each file is referred to by its name. The file is named for the convenience of the users and when a file is named, it becomes independent of the user and the process.

| | |
|---|---|
| (a) Name | (b) Type |
| (c) Location | (d) Size |
| (e) Protection | (f) Time and date |

## File Operations

The operating system provides system call to create, read, write, delete and truncate files. The common files operations are as follows :

### (i) File Creation

For a file to be created firstly, a space must be found for a file, in the file system and secondary a new entry must be made in the directory for the new file.

### (ii) Writing a File

In order to write a file, a system call is made which specify the file name and the information to be written in a file.

### (iii) Reading a File

To read from a file, we use a system call that specifies the name of the file and where in the memory, the next block of the file should be put and again for the associated directory entry.

- Total head

### (iv) Repositioning within a File

Repositioning within a file does not need to invoke any actual I/O. This repositioning within a file is also known as file seek.

### (v) File Deletion

The directory for the named file is searched in order to delete a file and when the associated directory entry is found, all the file space is released and the directory entry is removed.

### (vi) Truncating a File

Truncating a file is used when the user wants the attributes of the file to remain the same but wants to erase the contents of the file. There is no use of deleting a file and recreating it rather this function allow all attributes to remain unchanged but for the file to reset to length zero.

## Disk Scheduling

- *The operating system is responsible for using hardware efficiently; for the disk drives, this means having a fast access time and disk bandwidth.*
- *Access time has two major components–*
  - *→ Seek time is the average time taken by head to move from one track to another.*
  - *→ Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head. It is not fixed so we take average value.*

$$\text{Rotational latency} = \frac{\text{One complete revolution time}}{2}$$

### 1. FCFS Scheduling

- Process request sequentially.
- Fair to all processes.
- Approaches random scheduling in performance, if there are many processes.

  Queue = 98, 183, 37, 122, 14, 124, 65, 67

  Head starts at 53.

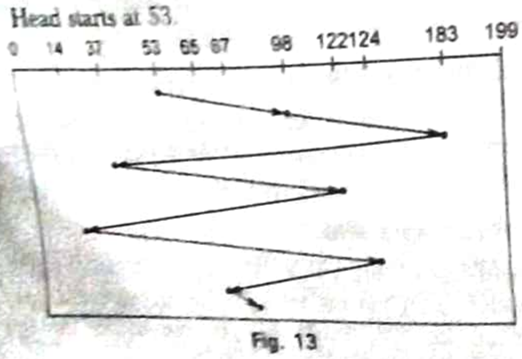


Fig. 13

### 2. Shortest Seek Time First (SSTF) Algorithm

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- It is not an optimal algorithm but its improvement over FCFC.
- Queue = 98, 183, 37, 122, 14, 124, 65, 67

  Head starts at 53.

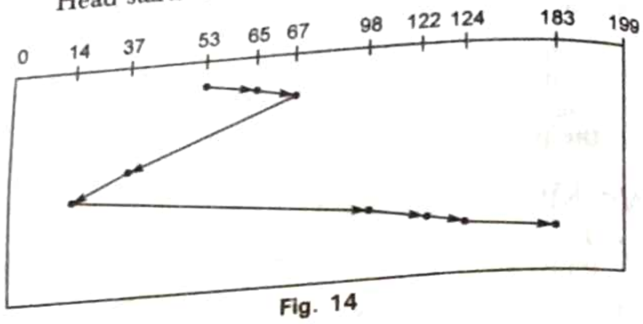


Fig. 14

- Total head movement are 208 cylinders.

### 3. SCAN Algorithm

- The disk arm starts at one end of the disk and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the elevator algorithm.

  Queue : 98, 183, 37, 122, 14, 124, 63, 67

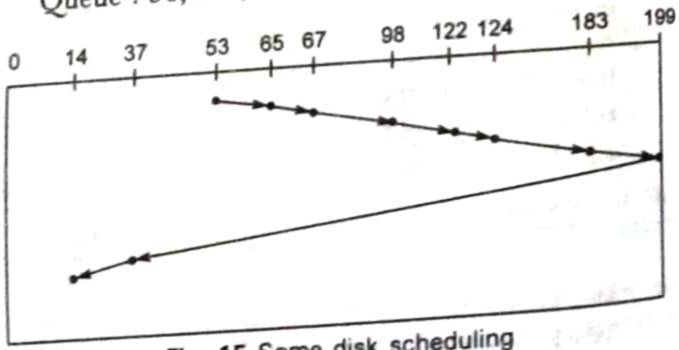


Fig. 15 Some disk scheduling

•