Unit VI SYSTEMS SOFTWARE AND ITS IMPORTANCE

Need of System Software, study of various components of system software.

Assemblers : Elements of Assembly Language Programming, A simple Assembly Scheme and pass structure of Assemblers.

Introduction to compilers : Phase structure of Compiler and entire compilation process. Introduction to Macro processors, Macro Definition and call, Macro Expansion Loaders and Linkers. General Loader Scheme, Subroutine Linkages, Relocation and linking Linkages, Relocation and linking. **(Chapter - 6)**

TABLE OF CONTENTS

Unit III

Chapter - 3 Concurrency Control (3 - 1) to (3 - 41)

3.1 Process / Thread Synchronization and Mutual Exclusion.....3 - 1
3.2 Mutual Exclusion : S/W .....3 - 4
3.3 Mutual Exclusion : H/W Approach .....3 - 6
3.4 Semaphore .....3 - 8
3.5 Monitor.....3 - 12
3.6 Classical Synchronization Problems .....3 - 14
3.7 Inter-Process Communication (Pipes, Shared Memory) .....3 - 21
3.8 Deadlock.....3 - 23
3.9 Deadlock Prevention .....3 - 28
3.10 Deadlock Avoidance .....3 - 29
3.11 Deadlock Detection and Recovery .....3 - 36
3.12 Example : Dining Philosophers Problem.....3 - 38

Unit IV

Chapter - 4 Memory Management (4 - 1) to (4 - 33)

4.1 Memory Management Concept and Requirements .....4 - 1
4.2 Memory Partitioning : Fixed Partitioning and Dynamic Partitioning.....4 - 3

4.3 Buddy System ..... 4 - 10
4.4 Paging ..... 4 - 12
4.5 Segmentation ..... 4 - 19
4.6 Virtual Memory ..... 4 - 23
4.7 Demand Paging ..... 4 - 25
4.8 Page Replacement ..... 4 - 27
4.9 Allocation of Frames, Thrashing ..... 4 - 32

Unit V

Chapter - 5 Input / Output and File Management (5 - 1) to (5 - 31)

5.1 I/O Devices, Organization of the I/O Function ..... 5 - 1
5.2 I/O Buffering ..... 5 - 3
5.3 Disk Scheduling ..... 5 - 6
5.4 Overview of File Management ..... 5 - 15
5.5 File Organization and Access ..... 5 - 19
5.6 File Directories ..... 5 - 22
5.7 File Sharing ..... 5 - 24
5.8 Record Blocking ..... 5 - 25
5.9 Secondary Storage Management ..... 5 - 26

Unit VI

Chapter - 6 Systems Software and its Importance (6 - 1) to (6 - 24)

6.1 Study of Various Components of System Software ..... 6 - 1

6.2 Assemblers ..... 6 - 2
6.3 Simple Assembly Scheme ..... 6 - 4
6.4 Pass Structure of Assembler ..... 6 - 5
6.5 Design of Two Pass Assembler ..... 6 - 7
6.6 Introduction to Macro Processors ..... 6 - 11
6.7 Macro Expansion ..... 6 - 11
6.8 Introduction to Compilers ..... 6 - 14
6.9 Phases of Compiler ..... 6 - 14
6.10 Loaders ..... 6 - 20
6.11 General Loader Scheme ..... 6 - 21
6.12 Subroutine Linkages ..... 6 - 21
6.13 Linkers ..... 6 - 22
6.14 Relocation and Linking ..... 6 - 23

Solved Model Question Papers (M - 1) to (M - 5)

Unit III

3 Concurrency Control

3.1 : Process / Thread Synchronization and Mutual Exclusion

Q.1 Explain the principle of concurrency.

Ans. : • Concurrent access to shared data may result in data inconsistency. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

- Concurrency arises in the same way at different levels of execution streams. Following are the example of concurrency in different types of operating systems :
  1. **Concurrenrrency in multiprogramming :** An interaction between multiple processes running on one CPU.
  2. **Concucy in multithreading :** An interaction between multiple threads running in one process
  3. **Concurrency in multiprocessors :** An interaction between multiple CPUs running multiple processes or threads.
  4. **Concurrency in multi-computers :** An interaction between multiple computers running distributed processes or threads.
- Java is a concurrent programming language.
- Process synchronization is required in uni-processor system, multiprocessor system and network. If more than one thread exists in a system at the same time, then the threads are said to be concurrent. Synchronization problems can occur whenever two or more concurrent processes use any shared resource.
- Cooperating process share the logical address space.

Q.2 Explain the following terms :
i) Critical section ii) Race condition

☞ [SPPU : Dec.-18,19, End Sem, Marks 5]

Ans. : i) Critical section :

- A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process may be in its critical section. The critical section problem is to ensure that **only one process at a time is allowed** to be operating in its critical section.

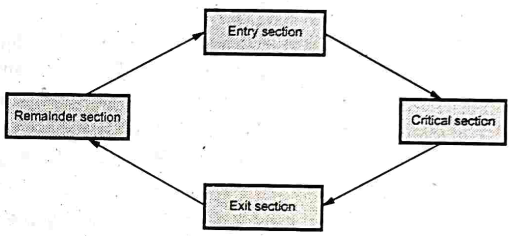


Fig. Q.2.1 Critical section

- Critical section means, process may change some common variable, writing files, updating memory location, updating a process table etc. When process is accessing shared modifiable data, it is said to be in a critical section.
- Each process takes permission from operating system to enter into the critical section. Structure of process is as follows :
  1. Entry section 2. Remainder section 3. Exit section
- **Entry section :** It is block of code executed in preparation for entering critical section.
- **Exit section :** The code executed upon leaving the critical section.
- **Remainder section :** Rest of the code is remainder section.
- Each process cycles through remainder, entry, critical, exit sections in this order.

ii) Race condition :

- **Race condition** occurs when two or more operations occur in an undefined manner. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called **race condition.**
- There is a "race condition" if the outcome depends on the order of the execution. The outcome depends on the CPU scheduling or "interleaving" of the threads.
- **Race condition** should be avoided because they can cause fine errors in applications and are difficult to debug.
- In banking system **balance** is a shared variable. After depositing the amount, it update by bank employee **balance = balance + amount.** At the same time, some amount is transfer then it becomes **balance = balance – amount.** These two tasks are in a race to write variable **balance.** In this the process that updates last determines the final value of balance.
- The concurrent execution of two processes is not guaranteed to be determinate, since different executions of the same program on the same data may not produce the same result.

Q.3 List the requirements of mutual exclusion.

☞ [SPPU : May-16, Dec.-17, End Sem, Marks 5]

Ans. : Requirements of mutual exclusion

1. At any time, only one process is allowed to enter in its critical section.
2. Solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about relative speeds of asynchronous concurrent processes.
5. A process cannot prevent any other process for entering into critical section.
6. A process must not be indefinitely postponed from entering its critical section

Q.4 Explain the following terms :
i) Mutual exclusion ii) Synchronization iii) Race condition

☞ [SPPU : Oct.-18, In Sem, Marks 6]

Ans. : i) Mutual exclusion :

- Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections
- Requirement of mutual exclusion is that, when process P1 is accessing a shared resource R1 then on other process should be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers and shared data structures.

ii) Synchronization : Synchronization means sharing system resources by processes in such way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes. Process synchronization was introduced to handle problems that arose while multiple process executions.

iii) Race condition : Refer Q.2.

3.2 : Mutual Exclusion : S/W

Q.5 How livelock situation is avaoided by using Dekker's solution ?

Ans. : • Dekker's solution avoids a livelock situation.

- Dekker's algorithm was the first correct solution to the critical section problem for two processes/threads. It uses an array of Boolean values and an integer variable.
- Dekker's algorithm is purely software solution with no special purpose hardware instruction. It uses flag to indicate a process desire to enter its critical section

- If two processes try to enter their critical sections at the same time, variable "turn" indicates which of the processes should be allowed to enter.
- It allows two threads to share a single-use resource without conflict, using only shared memory for communication.
- Processes are loops indefinitely, repeating entering and re-entering its critical section. Here flag and turn are the two global variables.
- Variable flag indicates the position of the process with respect to mutual exclusion. The turn variable is used for resolving the conflicts.
- Algorithm is as follows :

```
int c1=1;
int c2=1;
int turn=1;

process P1
while (true) {
 nonCriticalSection1;
 c1=0;
 while (c2!=1)
   if (turn==2){
     c1=1;
     while (turn!=1) {}
     c1=0;
   }
 criticalSection1;
 c1=1; turn=2;
}
end P1;

process P2
while (true) {
 nonCriticalSection2;
 c2=0;
 while (c1!=1)
   if (turn==1){
     c2=1;
     while (turn!=2) {}
     c2=0;
   }
 criticalSection2;
 c2=1; turn=1;
}
end P2;
```

- Here, if P1 finds c2 = 0, it defers to P2 only if turn = 2; otherwise, it simply waits for c2 to become 1 before entering its critical section.
- Process P2, which is also trying to enter its critical section at the same time, is forced to defer to P1 only if turn = 1.
  - Dekker's algorithm is correct and it satisfies mutual exclusion and it is free from deadlock and starvation
  - **Limitation of Dekker's Algorithm :**
    1. Dekker's algorithm does not provide strict alternation.
    2. Dekker's algorithm will not work with many modern CPUs.

    3. This algorithm won't work on Symmetric Multi-Processors (SMP) CPUs.
    4. Additionally, Dekker's algorithm can fail regardless of platform due to many optimizing compilers. Compiler may remove writes to flag since never accessed in loop. Creating an infinite loop.

Q.6 Write short note on : Peterson's solution.

Ans. : • Peterson's solution is software based solution for critical section problem. This algorithm will not work correctly on the modern computer architecture. It is simpler algorithm for two process mutual exclusion with busy waiting.

- This algorithm is less complicated than Dekker's algorithm.
- Peterson's algorithm solves critical section problem of two processes only. It does not require any special hardware. Two processes alternates execution between their critical section and remainder sections.
- Deadlock and indefinite postponement are impossible in Peterson's algorithm as long as no process or thread terminates unexpectedly.
- For indefinite postponement to occur, one process would have to be able to continually complete and reenter its critical section while the other process busy waited.

3.3 : Mutual Exclusion : H/W Approach

Q.7 Explain interrupt disabling concept of mutual exclusion.

Ans. : • On uniprocessor system, parallel execution of process is not possible. But they can only be interleaved. To guarantee mutual exclusion, it is sufficient to prevent a process from being interrupted.

- A process can prevent the operating system from executing by not issuing any supervisor call instructions and by disabling interrupts. Access to a critical section can be controlled by calling those primitives.

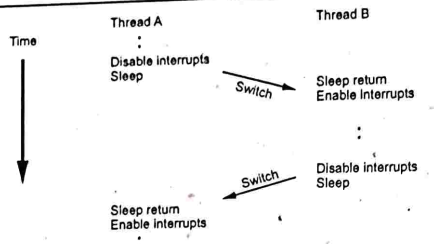


Fig. Q.7.1

- Use interrupts
  a) Implement preemptive CPU scheduling
  b) Internal events to relinquish the CPU
  c) External events to reschedule the CPU
- Disable interrupts
  a) Introduce uninterruptible code regions
  b) Think sequentially most of the time
  c) Delay handling of external events
- Mutual exclusion is guaranteed because the critical section cannot be interrupted

Disadvantages

1. It works only in a single processor environment.
2. Interrupts can be lost if not serviced promptly.
3. A process waiting to enter its critical section could suffer from starvation.

Q.8 What are the advantages and disadvantages of Test and Set operations?

Ans. : Advantages :

1. Simple to implement.
2. It works well for a small number of processes.
3. It can be used to support multiple critical sections.

Disadvantages :

1. It suffers from starvation.
2. There is possibility of busy waiting.
3. There may be deadlock.

3.4 : Semaphore

Q.9 Explain with definition, the concept of general and binary semaphore. [SPPU : May-17, Dec-17, End Sem, Marks 5]

Ans. : Concept of general semaphore :

- Semaphore is described by Dijkstra. Semaphore is a nonnegative integer variable that is used as a flag. Semaphore is an operating system abstract data type. It takes only integer value. Its used to solve critical section problem.
- Dijkstra introduced two operations (P and V) to operate on semaphore to solve process synchronization problem. A process calls the P operation when it wants to enter its critical section and calls V operation when it wants to exit its critical section. The P operation is called as wait operation and V operation is called as signal operation.
- A wait operation on a semaphore decease its value by one.

```
waits : while S < 0
        do loops;
S:= S – 1;
```

- A signal operation increments its value :

```
signal:
S:= S + 1;
```

- A proper semaphore implementation requires that P and V be indivisible operations. A semaphore operation is atomic. This may be possible by taking hardware support. The operation P and V are executed by the operating system in response to calls issued by any one process naming a semaphore as parameter.
- There is no guarantee that no two processes can execute wait and signal operations on the same semaphore at the same time.

Binary semaphore :

- Binary semaphore is also known as mutex locks. It deals with the critical section for multiple processes. Binary semaphore value is only between 0 and 1.
  1. The semaphore value is restricted to 0 and 1.
  2. P succeeds only when the semaphore value is 1.
  3. V does not change the semaphore value when it is 1.
- Binary semaphores are easier to implement than general semaphores

```
var
    mutex=1:binary-semaphore;
    delay=0:binary-semaphore;
    C={initvalue}:integer;
Procedure Wait()
      begin
    wait(mutex);
    C:=C-1;
    If C < 0 then begin
          signal(mutex);
          wait(delay);
          end
      else
                signal(mutex);
      end
Procedure Signal()
      begin
      wait(mutex);
      C:=C+1;
      If C <= 0 then
                signal(delay)
signal(mutex)
end
```


Q.10 List the properties of semaphore.

Ans. : Properties of semaphore :

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Semaphore acquire many resources simultaneously.
5. Can have many different critical sections with different semaphores.

Q.11 Explain the following functions (along with parameters passed) with reference to semaphore programming in 'C'.

i) sem_post( ) ii) sem_wait( ) ☞ [SPPU : Dec.-17, End Sem, Marks 5]

Ans. : i) sem_post ( )

Syntax :

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- The sem_post( ) function unlocks the specified semaphore by performing a semaphore unlock operation on that semaphore. When this operation results in a positive semaphore value, no threads were blocked waiting for the semaphore to be unlocked; the semaphore value is simply incremented.
- When this operation results in a semaphore value of zero, one of the threads waiting for the semaphore is allowed to return successfully from its invocation of the sem_wait( ) function. The sem_post( ) interface is reentrant with respect to signals and may be invoked from a signal-catching function.

ii) Sem_wait ( )

Syntax :

```
int sem_wait(sem_t *s) {
                wait until value of semaphore s is
               greater than 0
                decrement the value of semaphore s by 1
               }
```

- The sem_wait( ) function locks the semaphore referenced by sem by performing a semaphore lock operation on that semaphore.
- If the semaphore value is currently zero, then the calling thread will not return from the call to sem_wait( ) until it either locks the semaphore or the call is interrupted by a signal

Q.12 Explain busy waiting with appropriate example.

☞ [SPPU : Aug.-17, In Sem, Marks 4]

OR What is busy waiting with respect to process synchronization ? Explain how semaphore solves problem of synchronization.

☞ [SPPU : Oct-19, In Sem, Marks 6]

Ans. : • Busy waiting is a situation in which a process is blocked on a resource but does not yield the processor. A busy wait keeps the CPU busy in executing a process even as the process does nothing.

- Busy waiting is also called as spin waiting.
- Busy waiting waste CPU cycles that some other process might be able to use productively.
- To avoid busy waiting, a process waiting for critical section, they put into the blocked state. When process allowed entering into critical section, then process is kept in ready state.
- To overcome the need for bust waiting :

In wait( ) operation :

a. When process reads semaphore value and it is not positive value then process blocked itself for some time.

b. Semaphore keeps all blocked process in the waiting queue so process is kept in this queue.

c. CPU scheduler selects another process for execution.

In signal ( ) operation :

a. The wakup ( ) operation is used for restating the blocked processes. When any other process executes signal ( ) operation, then blocked process state changed from blocked state to ready state.

b. This process is kept in the ready queue.

3.5 : Monitor

Q.13 Explain monitor in brief.

☞ [SPPU : May-18,19, Dec-19, End Sem, Marks 5]

Ans. : • Monitor is an object that contains both data and procedures needed to perform allocation of a shared resource. It is a programming language construct that support both data access synchronization and control synchronization.

- Fig. Q.13.1 shows structure of monitor.

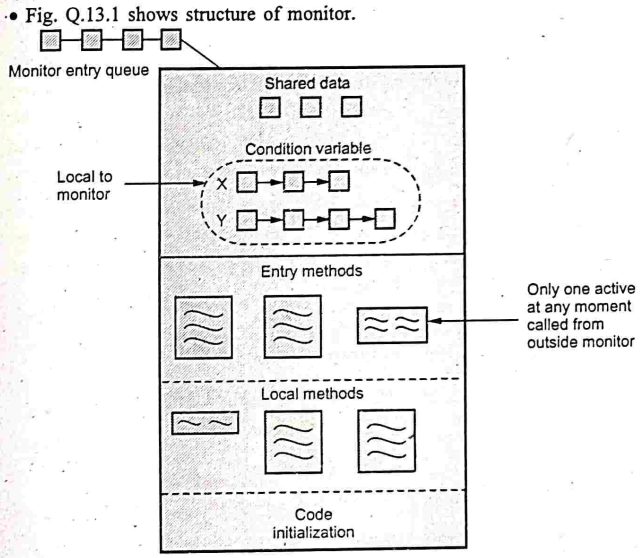


Fig. Q.13.1 Monitor structure

- Monitor is implemented in programming languages like Pascal, java and C++. Java makes extensive use of monitors to implement mutual exclusion.
- Monitor is an abstract data type for which only one process may be executing a procedure at any given time. Monitor is a collection of

procedure, variables and data structure. Data inside the monitor may be either global to all routines within the monitor or local to a specific routine.

- Monitor data is accessible only within the monitor. A shared data structure can be protected by placing it in a monitor. If the data in a monitor represents some resource, then the monitor provides a mutual exclusion facility for accessing the resource. A procedure defined within a monitor can access only those variable declared locally within the monitor and its formal parameters.
- When a process calls a monitor procedure, the first few instruction of the procedure will check to see if any other process is currently active within the monitor. If process is active then calling process will be suspended until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.
- Monitor supports synchronization by the use of condition variables that are contained within the monitor and accessible only within the monitor. Every conditional variable has an associated queue. Condition variables operates on two functions :

  cwait(condition variable)

  csignal (condition variable)

- **cwait :** It suspend execution of the calling process on condition.
- **csignal :** Resume execution of some process blocked after a cwait on the same condition.
- The cwait must come before the csignal.

Q.14 List the drawback of monitor.

Ans. :

1. Major weakness of monitors is the absence of concurrency if a monitor encapsulates the resource, since only one process can be active within a monitor at a time.
2. There is the possibility of deadlocks in the case of nested monitors calls.

3. Monitor concept is its lack of implementation most commonly used programming languages.
4. Monitors cannot easily be added if they are not natively supported by the language.

3.6 : Classical Synchronization Problems

Q.15 Explain reader writer problem.

Ans. : • When two types of processes need to access a shared resource such as a file or database. They called these processes reader and writers.

- Multiple readers can concurrently read from the data base. But when updating the database, there can only be one writer (i.e., no other writers and no readers either)
- More than one reader may read shared data (no writers). When a writer uses shared data, all other writers and readers must be excluded.
- Fig. Q.15.1 shows reader-writer problem.

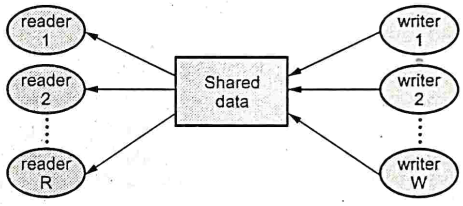


Fig. Q.15.1 Reader-writer problem

- For example in railway reservation system, readers are those who want train schedule information. They are called reader because readers do not change the content of the database. Many readers may access the database at once. There is no need to enforce mutual exclusion among readers.
- The writers are those who making reservations on a particular train. Writer can modify the database, so it must have exclusive access.

- When writer is active, no other readers or writers may be active. Therefore, it must enforce mutual exclusion if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set of processes. If a process want only to read the file, then it may share the file with any other process that also wants to read the file.
- We apply rules for access order :
  1. If a writer is waiting for readers to be finished, do not allow any more readers.
  2. If a reader is waiting for a writer to finish, give reader priority.
- If the writer wants to append the file, then no other process should have access to the file when the writer has access to it.
- If reader having higher priority than the writer, then there will be starvation with writers. For writer having higher priority than reader then starvation with readers.

Q.16 Write semaphore solution for dinning Philosophers problem.

☞ [SPPU : Dec-17, 19, End Sem, Marks 5]

Ans. : • There is one semaphore for each chopstick. In addition, a local two-phase prioritization scheme is used, under which philosophers defer to their neighbors who have declared themselves "hungry." All arithmetic is modulo 5.

```
system DINING_PHILOSOPHERS
VAR
me:    semaphore, initially 1;        /* for mutual exclusion */
s[5]:  semaphore s[5], initially 0;   /* for synchronization */
pflag[5]: {THINK, HUNGRY, EAT}, initially THINK;
                                      /* philosopher flag */

// As before, each philosopher is an endless cycle of thinking and
eating.
procedure philosopher(i)
```



```
 {
  while TRUE do
   {
    THINKING;
    take_chopsticks(i);
    EATING;
    drop_chopsticks(i);
   }
 }
// The take_chopsticks procedure involves checking the status of
neighboring philosophers and then declaring one's own intention to
eat. This is a two-phase protocol; first declaring the status HUNGRY,
then going on to EAT.
procedure take_chopsticks(i)
 {
  DOWN(me);          /* critical section */
  pflag[i] := HUNGRY;
  test[i];
  UP(me);            /* end critical section */
  DOWN(s[i])         /* Eat if enabled */
 }
void test(i)         /* Let phil[i] eat, if waiting */
 {
  if ( pflag[i] == HUNGRY
   && pflag[i-1] != EAT
   && pflag[i+1] != EAT)
    then
     {
      pflag[i] := EAT;
      UP(s[i])
     }
  }
```

```
// Once a philosopher finishes eating, all that remains is to relinquish
the resources, its two chopsticks and thereby release waiting
neighbors.
void drop_chopsticks(int i)
 {
  DOWN(me);          /* critical section */
  test(i-1);         /* Let phil. on left eat if possible */
  test(i+1);         /* Let phil. on rght eat if possible */
  UP(me);            /* up critical section */
 }
```

Q.17 Write a pseudo code for producer-consumer problem using semaphore. ☞ [SPPU : Aug-17, In Sem, Marks 6]

Ans. : • Solution to producer consumer problem by using binary semaphore :

```
void main ( )
{
    count = 0;
}
 int i;
binarysemaphore s = 1;
int producer ( )
{
   While (true)
{
        produce_data_item ( );
      semwaitB(s);
   append ( );
   count = count + 1;
   if ( count == 1)
   semSignalB(delay);
   semSignalB(s);
   }
}
 int consumer ( )
```



```
{
   int p;
semWaitB(delay);
while (true)
{
  semWaitB(s);
consume( );
count = count - 1;
p = count;
semSignalB(s);
consumedata( );
if ( p == 0)
semWaitB(delay);
}
}
```

Q.18 Write a semaphore solution for readers - writer's problem. ☞ [SPPU : May-18, Dec-18, End Sem, Marks 5]

Ans. : • When two types of processes need to access a shared resource such as a file or database. They called these processes reader and writers.

- The writers are those who making reservations on a particular train. Writer can modify the database, so it must have exclusive access. When writer is active, no other readers or writers may be active. Therefore, it must enforce mutual exclusion if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set of processes. If a process want only to read the file, then it may share the file with any other process that also wants to read the file. If the writer wants to append the file, then no other process should have access to the file when the writer has access to it.

```
int readcount = 0;
semaphore wsem = 1;
semaphore x = 1;
void main( )
        {
            int p = fork( );
            if(p) reader;          // assume multiple instances
            else writer;           // assume multiple instances
        }
```

```
void reader( )
    {
        while(1)
        {
            wait(x);
            readcount ++;
            if (readcount = = 1)
            wait(wsem);
            signal(x);
            doReading( );
        wait(x);
            readcount--;
        if (readcount==0)
            signal(wsem);
        signal(x);
        }
    }
```

```
void writer( ) {
    while(1) {
        wait(wsem)
        doWriting();
        signal(wsem)
    }
}
```

Q.19 What is critical section ? Give semaphore solution for producer - consumer problem. [SPPU : May-19, End Sem, Marks 5]

Ans. : • A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process may be in its critical section. The critical section problem is to

ensure that only one process at a time is allowed to be operating in its critical section.

- Producer - consumer problem is example classic problems of synchronization. Producer process produce data item that consumer process consumes later.
- Buffer is used between producer and consumer. Buffer size may be fixed or variable. The producer portion of the application generates data and stores it in a buffer and the consumer reads data from the buffer.
- In order to synchronize these processes, both procedure and consumer are blocked on some condition. The producer is blocked when the buffer is full and the consumer is blocked when the buffer is empty.

1. Code for producer :

```
producer (void)
{
    int item;
    while (TRUE)
    {
        produce item (& item);
        produce_if (counter = = N)
            sleep ( );
            enter_item (item);
            counter = counter + 1;
            if (counter = = 1)
            wakeup (consumer)
        }
}
```

2. Code for consumer process

```
consumer (void)
{
    int item;
    while (TRUE)
    {
        if (count = = 0)
        sleep ();
        remove_item (& item);
        counter = counter – 1;
        if (count = = N – 1)
        wakeup (producer);
        consume_item (item);
    }
}
```

3.7 : Inter-Process Communication (Pipes, Shared Memory)

Q.20 Enlist and explain different IPC mechanisms.

☞ [SPPU : May-18, Dec-18, End Sem, Marks 5]

Ans. : Interprocess Communication provides a mechanism to exchange data and information across multiple processes, which might be on single or multiple computers connected by a network.

- Five types of inter-process communication are as follows :
  1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
  2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
  3. Pipes permit sequential communication from one process to a related process.
  4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
  5. Sockets support communication between unrelated processes even on different computers.

1. Shared Memory :

- A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.

2. Memory Mapped :

- A memory-mapped file contains the contents of a file in virtual memory. This mapping between a file and memory space enables an application, including multiple processes, to modify the file by reading and writing directly to the memory.

3. Pipes :

- A pipe is a unidirectional, first-in first-out, unstructured data stream of fixed maximum size. Writers add data to the end of the pipe; readers retrieve data from the front of the pipe.
- Once read, the data is removed from the pipe and is unavailable to other readers. A pipe provides a simple flow control mechanism.

4. FIFO :

- Used to communicate between two processes that are not related. Full-duplex method - Process P1 is able to communicate with Process P2 and vice versa.

5. Socket :

- A socket is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets, one for each process. A socket is made up of an IP address concatenated with a port number. In general, sockets use a client-server architecture.

Q.21 Explain difference between named pipe and unnamed pipe.

Ans. : Difference between named pipe and unnamed pipe :

- A major difference between these IPC mechanism is the related processes are allowed to communicate using unnamed pipes but the unrelated processes are also communicate in named pipes.
- An unnamed pipe is only used for communication between a child and it's parent process, while a named pipe can be used for communication between two unnamed process as well.
- An unnamed pipe is a circular memory buffer that can be used to communicate between related processes. Named pipe is like FIFO.

Q.22 Explain the features of message passing system.

Ans. : Features of message passing

1. **Simplicity :** Message passing system should be simple and easy to use. It should be possible to communicate with old and new applications.
2. **Uniform semantics :** Message passing is used for two types of IPC.

a. **Local communication :** Communicating processes are on the same node.

b. **Remote communication :** Communicating processes are on the different nodes.

3. **Efficiency :** IPC become so expansive if message passing system is not effective. Users try avoiding to IPC for their applications. Message passing system will become more efficient if we try to avoid more message exchanges during communication process. For examples :

a. Avoiding the costs of establishing and terminating connection.

b. Minimizing the costs of maintaining the connections.

c. Piggybacking of acknowledgement.

4. **Reliability :** Distributed systems are prone to different catastrophic events such as node crashes or physical link failures. Loss of message because of communication link fails. To handle the loss messages, we required acknowledgement and retransmission policy. Duplicate message is one of the major problems. This happens because of timeouts or events of failures.

5. **Correctness :** Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows :

i. **Atomicity :** Every message sent to a group of receivers will be delivered to either all of them or none of them.

ii. **Ordered delivery :** Messages arrive to all receivers in an order acceptable to the application.

iii. **Survivability :** Messages will be correctly delivered despite partial failures of processes, machines, or communication links.

6. **Security :** Message passing system must provide a secure end to end communication.

7. **Portability :** Message passing system should itself be portable.

3.8 : Deadlock

Q.23 What are deadlocks ? Explain the different methods for dealing with the deadlocks.

Ans. : • Deadlock can be defined as the permanent blocking of a set of processes that either complete for system resources. Deadlock can occur on sharable resources.

- A process is in deadlock state if it was waiting for a particular event that will not occur. In a system deadlock, one or more processes are deadlocked.
- In resource deadlocks, processes can simultaneously wait for several resources and cannot proceed until they have acquired all those resources.
- In communication deadlocks, processes wait to communicate with other processes among a set of processes. A waiting process can unblock on receiving a communication from one of these processes.

Methods for Handling Deadlocks

1. Ensure that the system will never enter a deadlock state.
2. Allow the system to enter a deadlock state and then recover.
3. Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Q.24 Explain the necessary and sufficient conditions for the occurrence of a deadlock. ☞ [SPPU: Aug-17, Oct-18, In Sem, Marks 4]

Ans. : • Following four conditions are necessary for deadlock to exist.

1. Mutual exclusion 2. Hold and wait

3. No preemption 4. Circular wait

1. **Mutual exclusion :** A resource may be acquired exclusively by only one process at a time.
2. **Hold and wait :** Processes currently holding resources that were granted earlier can request new resources.
3. **No preemption :** Once a process has obtained a resource, the system cannot remove it from the process control until the process has finished using the resource.
4. **Circular wait :** A circular chain of hold and wait condition exists in the system.

- All four of these conditions must be present for a resource deadlock to occur. If one of them is absent, no resource deadlock is possible.

Q.25 Differentiate between the following: deadlock and starvation.

Ans. : • **Deadlocks :** A group of processes are waiting for resources held by others in the group. None of them will ever make progress.

- **Starvation :** A process may wait indefinitely because other process keep coming in and getting the requested resources before this process does.
- In a deadlock situation, none of the involved processes can possibly make progress. In a starvation situation, a process is ready to execute, but it is not being allowed to execute.

Q.26 Explain with an appropriate example, how resource allocation graph determines a deadlock.

☞ [SPPU : May-18, Dec-18, 19, End Sem, Marks 5]

Ans. : Resource allocation graphs

- Resource allocation graph is introduced by Holt. It is a directed graph that depicts a state of the system of resources and processes.
- Process and resource are represented by node in directed graph. Graph consists of a set of vertices (V) and set of edges (E).

Fig. Q.26.1 Process

- A process node is graphically represented by circle and shown in Fig. Q.26.1.
- A resource node is graphically represented by a rectangle and represents one type of resources. Fig. Q.26.2 shows resources node.

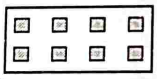


Fig. Q.26.3 Resource with 8 Instances

- The number of bullet symbols in a resource node indicates how many units of that resource class exist in the system.
- An arrow from the process to resource indicates the process is requesting the resource. An arrow from resource to process shows an instance of the resource has been allocated to the process.
- Claim edge P → R indicated that process P may request resource R in the future ; represented by a dashed line. Claim edge converts to request edge when a process requests a resource. Fig. Q.26.3 shows request and claim edge.

(a) Request edge (b) Claim edge

Fig. Q.26.3 Edges

- Request edge converted to an assignment edge when the resource is allocated to the process. When a resource is released by a process, assignment edge reconverts to a claim edge.
- Fig. Q.26.4 shows a resource allocation graph. System consists of process and resources.

Process : P_1, P_2

Resource : R_1, R_2

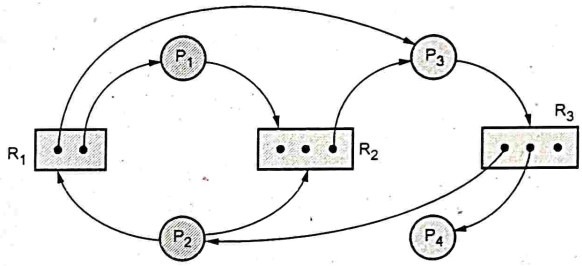


Fig. Q.26.4 Resource allocation graph

Resource	Number of instance
R_1	2
R_2	3
R_3	3

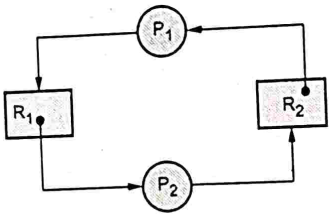


Fig. Q.26.5 Circular wait with deadlock

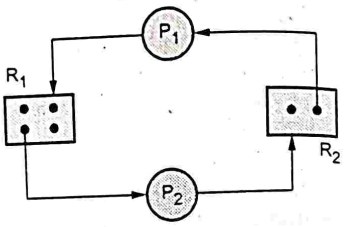


Fig. Q.26.6 Circular wait but no deadlock

Fig. Q.26.6 shows the resource allocation graph. System consists of four processes (P_1, P_2, P_3, P_4) and four resources (R_1, R_2, R_3, R_4)

Resource	Number of instance
R_1	2
R_2	2
R_3	3
R_4	6
R_5	3

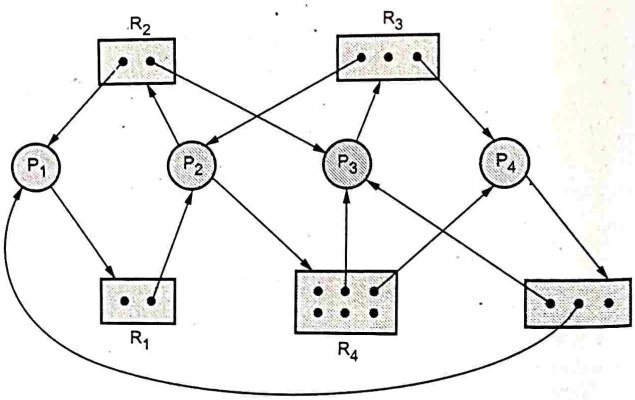


Fig. Q.26.7 Resource allocation graph

- Resource managers and other operating system processes can be involved in a deadlock. Deadlock is a global condition rather than a local one.
- An individual program cannot generally detect a deadlock. It is blocked and unable to use the processor to do any work. Deadlock detection must be handled by the operating system.

Q.27 How resource allocation graph helps to deadlock ? Write the necessary conditions of deeadlock to be occurred.

☞ [SPPU : Oct.-19, In Sem, Marks 4]

Ans. : Refer Q.26.

3.9 : Deadlock Prevention

Q.28 Explain deadlock prevention techniques with example.

☞ [SPPU : May-16, End Sem, Marks 5]

Ans. : • To prevent a deadlock, the OS must eliminate one of the four necessary conditions.

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

1. **Mutual exclusion :** It is necessary in any computer system because some resources (memory, CPU) must be exclusively allocated to one user at a time. No other process can use a resource while it is allocated to a process.
2. **Hold and wait :** If a process holding certain resources is denied a further request, it must release its original resources and if required request them again.
3. **No preemption :** It could be bypassed by allowing the operating system to deallocate resources from process.
4. **Circular wait :** Circular wait can be bypassed if the operating system prevents the formation of a circle.

- A deadlock is possible only if all four of these conditions simultaneously hold in the system.
- Prevention strategies ensure that at least one of the conditions is always false.

Q.29 How can the hold-and wait condition be prevented ?

Ans. : To prevent hold-and wait condition, processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

- There are several possibilities for this :
  1. Require that all processes request all resources at one time. This can be wasteful of system resources if a process needs one resource early in its execution and doesn't need some other resource until much later.
  2. Require that processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request. This can be a problem if a process has partially completed an operation using a resource and then fails to get it re-allocated after releasing it.
- Either of the methods described above can lead to starvation if a process requires one or more popular resources.

3.10 : Deadlock Avoidance

Q.30 Consider the following snapshot of a system. Answer the following questions using Banker's algorithm.

	Allocation	Maximum	Available
	A B C D	A B C D	A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

i) What are the contents of Need matrix ?
ii) Is the system in a safe state ?

☞ [SPPU : Aug-17, Oct-19, In Sem, Marks 6]

Ans. : i) Content of need matrix is

Process	R_1	R_2	R_3	R_4
P_1	0	0	0	0
P_2	0	7	5	0
P_3	1	0	0	2
P_4	0	0	2	0
P_5	0	6	4	2

ii) Safe state

a) Need of P_1 = (0, 0, 0, 0)

Available = (1, 5, 2, 0)

Need < Available

(Required resources are allocated to P_1)

∴New available = Allocation + Available

= (0, 0, 1, 2) + (1, 5, 2, 0)

= (1, 5, 3, 2)

b) For P_2 process

Need = (0, 7, 5, 0)

Available = (1, 5, 3, 2)

Need > Available

(Request is not granted)

New available = Available = (1, 5, 3, 2)

c) For P_3 process

Need = (1, 0, 0, 2)

Available = (1, 5, 3, 2)

Need < Available

(Request is granted)

New available = Allocation + Available

= (1, 3, 5, 4) + (1, 5, 3, 2)

= (2, 8, 8, 6)

d) For process P_4

Need of P_4 = (0, 0, 2, 0)

Available = (2, 8, 8, 6)

Need < Available

New available = (2, 14, 11, 8)

e) Process P_5

Need of P_5 = (0, 6, 4, 2)

Available = (2, 14, 11, 8)

Need < Available

New available = (2, 14, 11, 8) + (0, 0, 1, 4)

= (2, 14, 12, 12)

f) Again for P_2 process

Need of P_2 = (0, 7, 5, 0), Available = (2, 14, 12, 12)

Need < Available

New available = (1, 0, 0, 0) + (2, 14, 12, 12)

= (3, 14, 12, 12)

So the safe sequence is P_1, P_3, P_4, P_5, P_2

Q.31 Consider following snapshot of system.

Process	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answer the following questions using bankers algorithm.

i) What is the content of matrix need ?

ii) Is the system in safe state ? What is safe sequence ?

☞ [SPPU : April-16, In Sem, Marks 5; April-17, In Sem, Marks 6]

OR Consider the following snapshot of the system.

	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answer the following questions using Banker's algorithm.

a) What are the values of Need Matrix ?

b) Is the system in the safe mode ? If yes, what is the safe sequence ?

c) If a request from process P1 arrives for (1, 0, 2), can the request be granted immediately ?

☞ [SPPU : May-17, End Sem, Marks 10]

Ans. : Safe sequence : Safe sequence is calculated as follows :

1) Need of each process is compared with available. If $need_i \le available_i$, then the resources are allocated to that process and process will release the resource.

2) If need is greater than available, next process need is taken for comparison.

3) In the above example, need of process P_1 is (7, 4, 3) and available is (3, 3, 2).

need ≥ available → False

So system will move for next process.

4) Need for process P_2 is (1, 2, 2) and available (3, 3, 2), so

need ≤ available (work)

(1, 2, 2) ≤ (3, 3, 2) = True

then Finish [i] = True

Request of P_2 is granted and processes P_2 is release the resource to the system.

Work : = Work + Allocation

Work : = (3, 3, 2) + (2, 0, 0)

: = (5, 3, 2)

This procedure is continued for all processes.

5) Next process P_3 need (6, 0, 0) is compared with new available (5, 3, 2).

Need > Available = False

(6 0 0) > (5 3 2)

6) Process P_4 need (0, 1, 1) is compared with available (5, 3, 2).

Need < Available

(0 1 1) < (5 3 2) = True

Available = Available + Allocation

= (5 3 2) + (2 1 1)

= (7 4 3) (New available)

7) Then process P_5 need (4 3 1) is compared with available (7 4 3)

Need < Available

(4 3 1) < (7 4 3) = True

Available = Available + Allocation

= (7 4 3) + (0 0 2) = (7 4 5) (New available)

8) One cycle is completed. Again system takes all remaining process in sequence. So process P_1 need (7 4 3) is compared with new available (7 4 5).

Need < Available = True

(7 4 3) < (7 4 5)

Available = Available + Allocation

= (7 4 5) + (0 1 0) = (7 5 5) (New available)

9) Process P_3 need is (6 0 0) is compared with new available (7 5 5).

∴ Need < Available = True

(6 0 0) < (7 5 5) = True

Available = Available + Allocation

= (7 5 5) + (3 0 2)

= (10 5 7) = (New available)

Safe sequence is < P_2 P_4 P_5 P_1 P_3 >

Q.32 What is banker's algorithm? Explain data structure used by banker's algorithm.

Ans. : • Banker's algorithm is the deadlock avoidance algorithm. Banker's is named because the algorithm is modeled after a banker who makes loads from a pool of capital and receives payments that are returned to that pool.

- Algorithm is check to see if granting the request leads to an unsafe state. If it does, the request is denied. If granting the request leads to a safe state, it is carried out.
- The Dijkstra proposed an algorithm to regulate resource allocation to avoid deadlocks. The banker's algorithm is the best known of the avoidance method.
- By using avoidance method, the system is always kept in a safe state.
- It is easy to check if a deadlock is created by granting a request, deadlock analysis method is used for this.
- Deadlock avoidance uses the worst-case analysis method to check for future deadlocks.
- **Safe state :** There is at least one sequence of resource allocations to processes that does not result in a deadlock.
- System is in a safe state only if there exist a safe sequence. A safe state is not a deadlocked state.
- Deadlocked state is an unsafe state. It does mean the system is in a deadlock.
- As long as the state is safe, the resource manager can be guaranteed to avoid a deadlock.
- Initially the system is in a safe state. When process requests a resource and that resource is available then the system must decide whether the resources can be allocated immediately or process must wait.

- If system remains in safe state after allocating resource then only OS allocates resources to process.
- Banker algorithm uses following data structures.

1. Allocation : Allocation is a table in which row represents process and column represents resources (R).

alloc [i, j] = Number of unit of resource R_j held by process P_i.

2. Max : Max be the maximum number of resources that process requires during its execution.

- **Need (claim) :** It is current claim of a process, where a process's claim is equal to its maximum need minus its current allocation.

Need = Max – Allocation

- **Available :** There will be number of resources still available for allocation. This is equivalent to the total number of resources minus the sum of the allocation to all processes in the system.

Available = Number of resources – Sum of the allocation to all process

$$= \text{Number of resources} - \sum_{i=1}^{n} \text{Allocation } (P_i)$$

- Each process cannot request more that the total number of resources in the system. Each process must also guarantee that once allocated a resource, the process will return that resource to the system within a finite time.

Q.33 List weakness of Bankers algorithm.

Ans. : Weakness of Banker's algorithm

1. It requires that there be a fixed number of resources to allocate.
2. The algorithm requires that users state their maximum needs (request) in advance.
3. Number of users must remain fixed.
4. The algorithm requires that the bankers grant all requests within a finite time.
5. Algorithm requires that process returns all resource within a finite time.

3.11 : Deadlock Detection and Recovery

Q.34 State and explain the deadlock recovery methods.

Ans. : • Once deadlock has been detected in the system, the deadlock must be broken by removing one or more of the four necessary conditions.

- Here one or more processes will have to be preempted, thus releasing their resources so that the other deadlocked processes can become unblocked.

1. Process termination

- Deadlock is removed by aborting a process. But aborting process is not easy. All deadlocked processes are aborted.
- Circular wait is eliminated by aborting one by one process. There will be lot of overhead. Deadlock detection algorithm must rerun after each process kill.
- Selection of process for aborting is difficult. Following parameters are considered :

1. Priority of the process.
2. What percentage the process finished its execution ?
3. Resource used by process.
4. Need of resources to complete process remaining operation.
5. How many processes will need to be terminated ?
6. Process type : Batch or interactive.

2. Resource preemption

- Some times, resource temporarily take away from its current process and allocate it to another process.
- For selecting victim, following factors are considered.
  1. Priority of the process. Higher priority process are usally not seleced.
  2. CPU time used by process. The process which is close to completion are usually not selected.
  3. The number of other process that would be affected if this process were selected as the victim.

3. Recovery through rollback

- When a process in a system terminates, the system performs a rollback by undoing every operation related to the terminated process.
- Checkpointing a process means that its state is written to a file so that it can be restarted later.
- Risk in this method is that the original deadlock may recover but the nondeterminancy of concurrent processing may ensure that this does not happen.

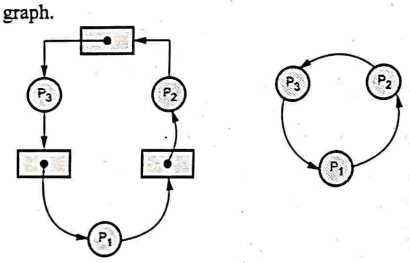
4. Starvation

- Starvation is one type situation in which a process waits for an event that might never occur in the system.
- Select the victim only for finite number of times. Use rollback method for selecting victim process.

Q.35 What is wait for graph ?

Ans. : • Any resource allocation graph with a single copy of resources can be transferred to a **wait for graph.**

- Fig. Q.35.1 shows resource allocation graph with corresponding wait for graph. The state of the system can be modeled by directed graph, called a wait for graph.



(a) Resource allocation graph (b) Wait for graph

Fig. Q.35.1

- Wait for graph is a graph where each node represents a process. An edge $P_i \rightarrow P_j$ means that process P_i is blocked and waiting for process P_j to release a resource.

- The wait for graph of a system is always smaller than the resource allocation graph of that same system. There is a deadlock in a system if and only if there is a loop in the wait for graph of that system.
- Deadlock detection invloves two issues :
  1. Maintenance of the wait for graph.
  2. Searching of the wait for graph for the presence of cycles.
- Deadlock detection requires overhead for run time cost of maintaining necessary information and executing the detection algorithm.
- For multiple instances of a resource type use an algorithm similar to banker's algorithm.

3.12 : Example : Dining Philosophers Problem

Q.36 Write a deadlock - free solution for dining philosopher's problem using semaphore.

☞ [SPPU : Oct-18, In Sem, Marks 6, May-19, End Sem, Marks 5]

Ans. : • There is one semaphore for each chopstick. In addition, a local two-phase prioritization scheme is used, under which philosophers defer to their neighbors who have declared themselves "hungry." All arithmetic is modulo 5.

```
system DINING_PHILOSOPHERS
VAR
me:    semaphore, initially 1;        /* for mutual exclusion */
s[5]:  semaphore s[5], initially 0;   /* for synchronization */
pflag[5]: {THINK, HUNGRY, EAT}, initially THINK;
                                      /* philosopher flag */

// As before, each philosopher is an endless cycle of thinking and
eating.
procedure philosopher(i)
 {
  while TRUE do
   {
    THINKING;
```

```
    take_chopsticks(i);
    EATING;
    drop_chopsticks(i);
  }
}
```

// The take_chopsticks procedure involves checking the status of neighboring philosophers and then declaring one's own intention to eat. This is a two-phase protocol; first declaring the status HUNGRY, then going on to EAT.

```
procedure take_chopsticks(i)
 {
  DOWN(me);          /* critical section */
  pflag[i] := HUNGRY;
  test[i];
  UP(me);            /* end critical section */
  DOWN(s[i])         /* Eat if enabled */
 }
void test(i)         /* Let phil[i] eat, if waiting */
 {
  if ( pflag[i] == HUNGRY
   && pflag[i-1] != EAT
   && pflag[i+1] != EAT)
    then
     {
      pflag[i] := EAT;
      UP(s[i])
     }
  }
```

// Once a philosopher finishes eating, all that remains is to relinquish the resources, its two chopsticks and thereby release waiting neighbors.

```
void drop_chopsticks(int i)
 {
  DOWN(me);          /* critical section */
```



```
  test(i-1);         /* Let phil. on left eat if possible */
  test(i+1);         /* Let phil. on rght eat if possible */
  UP(me);            /* up critical section */
 }
```

Q.37 Explain dining philosophers problem.

Ans. : • Dinning philosophers problem is one of classical process synchronization problem. Here five philosophers are seated around a circular table. They spend their lives in thinking and eating. Five plates with five forks are kept on the circular table.

- While philosophers think, they ignore the eating and do not require fork. When a philosopher decides to eat, then he/she must obtain a two fork, one from left side fork and another from right side fork. Philosophers can pick up only a single fork at one time.
- After consuming a food, the philosophers replace the fork and resumes thinking. A philosophers to the left or right of a dinning philosopher cannot eat while the dinning philosophers is eating, since forks are a shared resource.
- Fig. Q.37.1 shows seating arrangement of five philosophers.
- Consider the following code :

```
void dinning_Philosopher ( )
  {
     while (true)
     {
        thinking ( );
        eating ( );
     }

  }
void eating ( )
 {
  takeleftfork ( );
        takerightfork ( );
        eatfood(delay);
```

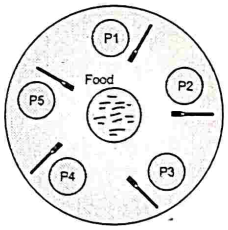


Fig. Q.37.1 Dinning philosophers

```
                putrighttfork ( );
    putleftfork ( );
    }
```

- Here philosophers operates asynchronously and concurrently, it is possible for each philosophers to enter into eating mode. The procedure takeleftfork ( ) waits until the specified fork is available and then seizes it. This is not possible because of the following reasons.
- Each philosopher will hold exactly one fork, and no forks will remain available on the table. The philosophers will all deadlock.
- To solve this problem, write some extra code in takerightfork ( ) procedure. User can specify the philosophers to put down the left fork if that philosophers cannot obtain the right fork.

Problem analysis :

- **Shared resource :** Here each fork is shared by two philosophers so we called it is a shared resource.
- **Race condition :** we do not want a philosopher to pick up a fork that has already been picked up by his neighbor. This is a race condition.
- **Solution :** we consider each fork as a shared item and protected by a mutex lock. Before eating, each philosopher first lock left fork and then right fork. If philosopher acquires both locks successful, then philosophers have two locks (two fork) so he can eat a food. After finishes easting, this philosopher releases both fork, and thinks.

END...

Unit IV

4 Memory Management

4.1 : Memory Management Concept and Requirements

Q.1 Explain primary memory requirement. Also list functions of memory management.

Ans. : Primary memory requirements

1. **Access time :** It should be as small as possible. This need influences both software and hardware design.
2. **Size :** Size must be as large as possible. It can accomodate many programs into memory.
3. **Cost :** Cost of the memory is less than the total cost of the computer.

Memory Management Function

1. Allocate primary memory space to processes.
2. Minimize access time.
3. Determining allocation policy for memory.
4. Deallocation technique and policy.

Q.2 Define the following :
Compile time, load time, execution time.

Ans. : • **Compile time :** Source program is translated at compile time to produce a relocatable object module. At compile time, the translator generates code to allocate storage for the variable. This storage address is used for code reference. Target address is unknown at compile time, it cannot be bound at compile time. Example of compile time binding is MS DOS.com programs.

- **Load time :** Compiler generates relocatable code if compile time binding is not performed. The loader modifies the addresses in the load

module at load time to produce the executable image stored in main memory. Final binding is delayed until program load time.

- **Execution time** : Memory address of the program is changed at execution time, then execution time binding is used. Binding is delayed until the run time of the program. Normally all operating system uses execution time binding. Special hardware is used for execution time binding.

Q.3 What are requirement for memory management ?

☞ [SPPU : May-16, End Sem, Marks 8]

Ans. : • Memory management requirements are Relocation, Protection, Sharing, Logical organization and Physical organization.

- **Relocation** : Programmer does not know where the program will be placed in memory when it is executed. While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated). Memory references must be translated in the code to actual physical memory address.
- The operating system need to know the location of :
  1. Process control information
  2. Execution stack
  3. Entry point to begin the execution of a program. 'Processor' must deal with memory references within the program.
- **Protection** : Memory protection is used to avoid interference between programs existing in main memory. The memory protection hardware compares every memory address used by the program with the contents of two registers (base and limit) to ensure that it lies with the allocated memory area. Multiple hardware memories are used to provide a larger address space.
- The simplest method of memory protection is adding two registers to the CPU. This works good for all memory is allocated contiguously. Non-contiguous memory is harder to protect.
- **Sharing** : Allow several processes to access the same portion of memory. Better to allow each process access to the same copy of the program rather than have their own separate copy.

- **Logical organization** : Main memory is organized as a linear or one-dimensional address space that consists of sequence of bytes or words. Secondary memory at its physical level is similarly organized. Most of the programs are organized into modules.
- **Physical organization** : Computer memory is organized into two levels :
  a) Main memory : Main memory is a volatile memory and it provides fast access at relatively high cost.
  b) Secondary memory : Secondary memory is a non-volatile memory and it is slower and cheaper than main memory

4.2 : Memory Partitioning : Fixed Partitioning and Dynamic Partitioning

Q.4 Explain fixed partition of memory.

Ans. : Contiguous Memory Allocation with Fixed Partitions

- In a fixed size partitioning of the main memory all partitions are of the same size. The operating system divides main memory into a number of fixed size partition. Each partition holds a single program.
- Fixed sized partitions are relatively simple to implement. Degree of multiprogramming depends on the number of partitions.
- Normally main memory is divided into two partitions :
  1. For resident program 2. For user processes
- Batch operating system uses the fixed size partition scheme. The operating system keeps the record of memory allocation and deallocation in the form of table. Initially all the memory is available for user processes.
- When the process arrives in the system and needs memory, operating system search for large hole for this process. Hole is one large block of free available memory. If any free hole is found, process is allocated to the free hole of memory as is needed.
- After allocating number of holes for the processes, a set of various size holes is scattered throughout memory at any given time. When a

process arrives and searches for (memory) set of holes. But the holes must be large enough to accomodate the process.

- Fixed sized partitions are simple to implement. Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- Fig. Q.4.1 shows an example of fixed partitioning of a memory.
- There are two difficulties with the use of equal size fixed partitions :
  1. A program may be too big to fit into a partition.
  2. Memory utilization is extremely inefficient.
- First difficulty is solved by using overlays. Overlays involves moving data and program segments in and out of memory.

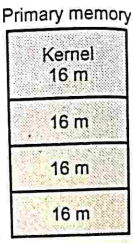


Fig. Q.4.1 Fixed size partition

Advantages and disadvantages of fixed partition size

1) Advantages

1. Simple to implement.
2. Does not require expertise to understand and use such a system.
3. Less overhead.

2) Disadvantages

1. Memory is not fully utilized. 2. Poor utilization of processors.
3. User's process being limited to the size of available main memory.
4. Requires contiguous loading of entire program.

Q.5 Explain dynamic partition of memory.

Ans. : Contiguous Memory Allocation with Variable Partitions

- The use of unequal size partitions provides a degree of flexibility to fixed partitioning. In dynamic partitioning, the partitions are of variable length and number.
- In noncontiguous memory allocation, a program is divided into blocks that the system may place in nonadjacent slots in main memory.
- This allocation method do not suffer from internal fragmentation, because a process partition is exactly the size of the process.

- Fig. Q.5.1 shows noncontiguous memory allocation method. Operating system maintain the table which contains the memory areas allocated to process and free memory. Memory management unit use this information for allocating processes.

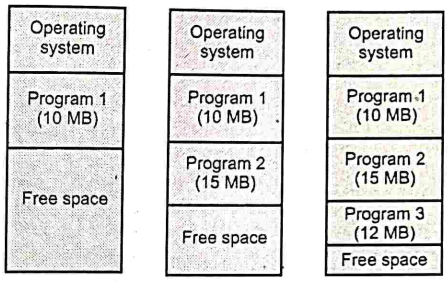


Fig. Q.5.1 (a) Variable size partition

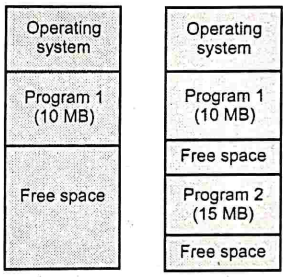


Fig. Q.5.1 (b) Noncontiguous memory allocation

- Table contains information about memory starting address for process/program and their size.
- CPU sends the logical address of the process to the MMU and the MMU uses the memory allocation information stored in the table for calculating logical address. We called this address as effective memory address of the data/instruction.

Q.6 Explain difference between contiguous and non-contiguous memory allocation.

Ans. : Difference between Contiguous and Noncontiguous Memory Allocation

Sr. No.	Contiguous allocation	Noncontiguous allocation
1.	Program execution take place without overhead.	Address translation is overhead.
2.	Swapped-in processes are placed in the original area.	Swapped-in processes can be placed any where in memory.
3.	Suffer from internal fragmentation.	Only paging, suffers from internal fragmentation.
4.	Allocates single area of memory for process.	Allocates more than one block of memory for process.
5.	Wastage of memory.	No wastage of memory.

Q.7 What is fragmentation? Explain types of fragmentation.

Ans. : • Fragmentation is a phenomenon in which storage space is used inefficiently, reducing capacity or performance and often both. The exact consequences of fragmentation depend on the specific system of storage allocation in use and the particular form of fragmentation.

- Fragmentation are of two types :
  1. Internal fragmentation.
  2. External fragmentation.
- In fragmentation, OS cannot use certain area of available memory.

Internal fragmentation

- There is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition is called as internal fragmentation.
- Fig. Q.7.1 shows an internal fragmentation.

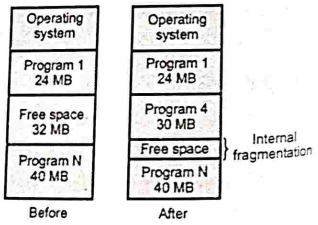


Fig. Q.7.1 Internal fragmentation

- To keep track of this free hole is overhead for system.

External fragmentation

- It occurs when enough total main memory space exists to satisfy a request, but it is not contiguous, storage is fragmentated into a large number of small holes.
- Fig. Q.7.2 shows external fragmentation.

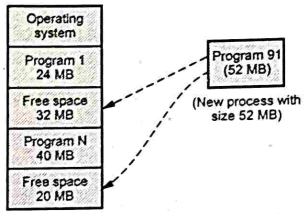


Fig. Q.7.2 External fragmentation

- Following are the solution for external fragmentation :
  1. Compaction
  2. Logical address space of a process/program

Q.8 Explain the difference between internal and external fragmentation.

Ans. :

Sr. No.	Internal fragmentation	External fragmentation
1.	Memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation.	External fragmentation exists when there is enough total memory space to satisfy a request, but available spaces are not contiguous.
2.	First-fit and best-fit memory allocation does not suffer from internal fragmentation.	First-fit and best-fit memory allocation suffers from external fragmentation.
3.	Systems with fixed sized allocation units, such as the single partitions scheme and paging suffer from internal fragmentation.	Systems with variable sized allocation units, such as the multiple partitions scheme and segmentation suffer from external fragmentation.

Q.9 Explain different ways to remove external fragmentation.

[SPPU : May-18, Nov-19, End Sem, Marks 6]

Ans. : • Compaction solves problem of external fragmentation.

- Operating system moves all the free holes to one side of main memory and creates large block of free size.
- It must be performed on each new allocation of process to memory or completion of process for memory. System must also maintain relocation information.
- All free blocks are brought together as one large block of free space. Compaction requires dynamic relocation.
- Fig. Q.9.1 shows compaction.

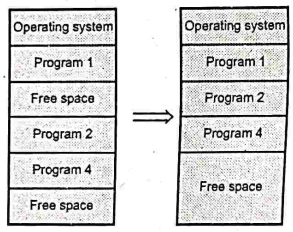


Fig. Q.9.1 Compaction

- Compaction has a cost and selection of an optimal compaction strategy is difficult. One method for compaction is swapping out those processes that are to be moved within the memory and swapping them into different memory locations.
- Compaction is not always possible. Compaction is time consuming method and wastage of the CPU time.

Garbage collection

- Some programs use dynamic data structures. These programs dynamically use and discard memory space. Technically, the deleted data items release memory locations.

- But, in practice the OS does not collect such free space immediately for allocation. This is because that affects performance. Such areas, therefore are called **garbage.**
- When such garbage exceeds a certain threshold the OS would not have enough memory available for any further allocation.

Q.10 Explain following terms in brief (any two) :

i) Internal fragmentation ii) External fragmentation iii) Compaction

[SPPU : May-19, Dec.-19 End Sem, Marks 4]

Ans. : i) Internal fragmentation : Refer Q.7.

ii) External fragmentation : Refer Q.7.

iii) Compaction : Refer Q.9.

Q.11 Free memory holes of sizes 15 K, 10 K, 5 K, 25 K, 30 K, 40 K are available. The processes of size 12 K, 2 K, 25 K, 20 K, are to be allocated. How processes are placed using first fit, best fit and worst fit partitioning algorithm. Calculate internal and external fragmentation. [SPPU : May-18, Dec-18, 19, End Sem, Marks 10]

Ans. : Free memory holes = 15 K, 10 K, 5 K, 25 K, 30 K, 40 K

Process size = 12 K, 2 K, 25 K, 20 K

First Fit : 12 K → 15 K, 2 K → 10 K, 25 K → 25 K, 20 K → 30 K

Best Fit : 12 K → 15 K, 2 K → 5 K, 25 K → 25 K, 20 K → 30 K

Worst Fit : 12 K → 40 K, 2 K → 30 K, 25 K → 28 K (40 K – 12 K), 20 K → 28 K (30 K – 2 K)

Q.12 Given memory partitions of 100 K, 500 K, 200 K, 300 K and 600 k (in-order), how would each of First-Fit, Best-Fit and Worst-Fit algorithms place processes of 212 K, 417 K, 112 K, 426 K ? Which algorithm makes the most efficient use of memory ?

[SPPU : May-17, End sem, Marks 8]

Ans. : First-fit :

212 K is put in 500 K partition

417 K is put in 600 K partition

112 K is put in 288 K partition

(new partition 288 K = 500 K – 212 K)

426 K must wait.

Best-fit :

212 K is put in 300 K partition

417 K is put in 500 K partition

112 K is put in 200 K partition

426 K is put in 600 K partition

Worst-fit :

212 K is put in 600 K partition

417 K is put in 500 K partition

112 K is put in 388 K partition (600 K – 212 K)

426 K must wait

In this example, Best-fit turns to be the best.

4.3 : Buddy System

Q.13 Write a short note on : Buddy system.

[SPPU : Dec-17,19, May-19, End Sem, Marks-4]

Ans. : • The buddy system is a simple dynamic storage allocation method. The buddy system is a memory allocation and management algorithm that manages memory in power of two increments. Linux operating system manages memory using buddy algorithm.

- Fig. Q.13.1 shows buddy system.
- The buddy system assumes that memory is of size 2^N for some integer N. Both free and reserved blocks will always be of size 2^k for $k < N$. At any given time, there might be both free and reserved blocks of various sizes.
- The buddy system keeps a separate list for free blocks of each size. There can be at most N such lists, because there can only be N distinct block sizes.
- Single allocation block to be split, to form two blocks half the size of the parent block. These two blocks are known as 'buddies'.

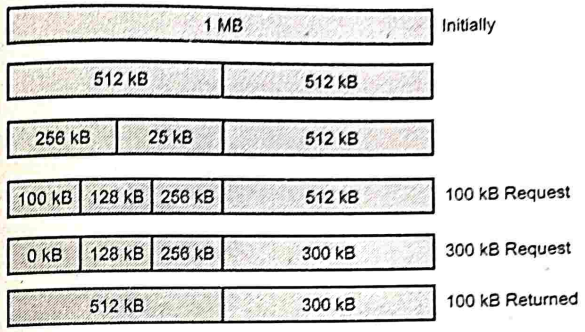


Fig. Q.13.1 : Buddy system

- Initially memory consists of a single contiguous area of 128 pages. When user send request for memory, it is first rounded up to a power of 2.
- All the lists are initially empty, except for the 1 MB list, which has one hole listed. All allocations are rounded up to a power of two. Suppose the request for memory is 75 K then allocations rounded up to 128 K, 13 K allocations rounded up to 16 K, etc.
- If a block of that size is free, it is taken; otherwise, the smallest free block larger than the desired size is found and split in two halves. This splitting continuous until the appropriate size is reached.
- When memory is deallocated, the buddy system groups contiguous free pages. When process free a block of memory, the memory manager checks the bitmap for checking block size. If adjacent block is also free, the memory manager combines the two blocks into a larger block.
- This method makes deallocation fast. If a block of size 2^K is free then the memory manager checks only the list of 2^K holes to merge them into a 2^{K+1} sized partition. Internal fragmentation is caused since memory requests are fitted in 2^K sized partitions.

Advantages

a) Easy to implement a buddy system

b) Allocates block of correct size

c) It is easy to merge adjacent holes

d) Fast to allocate memory and de-allocating memory

Disadvantages

a) It requires all allocation unit to be powers of two

b) It leads to internal fragmentation.

c) Merging of holes is recursive, so poor worst-case behavior.

4.4 : Paging

Q.14 What is paging ? Explain in details.

Ans. : • In paging, operating system divides each incoming programs into pages of equal size. The sections of a disk are called block or sectors. The sections of main memory are called page frames. One sector will hold one page of job instructions and fit into one page frame of memory.

- In paging, logical address space of a program can be noncontiguous. It solves external fragmentation problem.
- The relation between virtual addresses and physical memory addresses given by page table.

• Fixed sized blocks are called frames and breaking of logical memory into blocks of same size called pages.

• Memory manager prepares following things before executing a program :

1. Find out the number of pages in the program.
2. Free space in the main memory.
3. Loading of all the programs pages into memory.

• Pages are not loaded continuosly in main memory. Each page can be stored in any available page frame anywhere in main memory. Memory manager keeps the track of pages of program. Paging avoids external fragmentation and need for compaction.

• Memory manager uses three tables to keep track of process and memory.

1. **Job table :** It stores the size of the active job and memory location where its page table is stored.
2. **Page map table :** It contains vital information about each page. PMT contains apage number and corresponding page frame memory address. It includes only one entry per page. Page numbers are in sequential order.
3. **Memory map table :** MMT is used to store page frame location and its status.

• Page size depends upon underlying hardware. Operating system maintains a page table for each process. The page table shows the frame location for each page of a process. Fig. Q.14.1 shows the virtual address format for paging system.

Fig. Q.14.1 Virtual address format

• Processor hardware performs the logical to physical address translation. Logical address contains page number and offset. The physical address contains frame number and offset. Offset is a relative factor. It is used to locate that line within its page frame.

• Fig. Q.14.2 shows address translation.

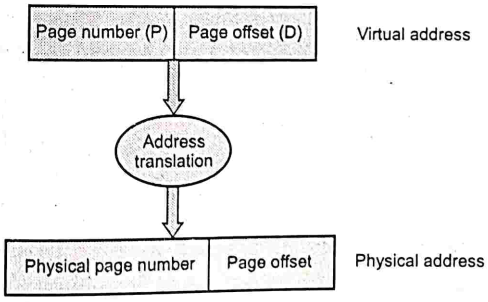


Fig. Q.14.2 Address translation

- Suppose a system uses n-bit for representing both physical and virtual address. The page number is represented by the most significant bit (n - m bits) and the displacement is represented by m bit.
- The table, which holds virtual address to physical address translations is called page table. As displacement is constant, so only translation of virtual page number to physical page is required.

Q.15 What are the common techniques for structuring the page table ? Explain at least three of these techniques.

☞ **[SPPU : Dec-17, End Sem, Marks 10]**

Ans. : Page Table Structure :

1) Multilevel or hierarchical page table :

- Page tables can consume a significant amount of memory.
- For example : A 32-bit virtual address space using 4 kB pages.

$$\text{Page size} = 2^{12} \text{ bytes}$$

$$\text{Space for page numbers} = 2^{32} - 2^{12} = 2^{20} \text{ bytes}$$

- So page table may consists of upto 1 million entries, one for each page, for a total address capacity of about four billion bytes:
- Hierarchical page table is also called as forward mapped page table. This approach is so effective that many modern operating systems employ it.
- In this method, each level containing a table that stores pointers to tables in the level below. Each table in the hierarchy is the size of one page. It enables the operating system to transfer page tables between main memory and secondary storage device easily.

For two levels of page table

- Virtual address contains page number and displacement into that page.

Virtual address (v) = (p, t, d)

where (p, t) = Page number

d = Displacement

- Here p is an index into the outer page table.
- Fig. Q.15.1 shows hierarchical page table.

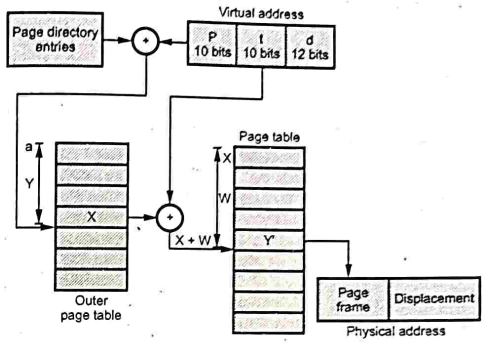


Fig. Q.15.1 Two level page table method

- Intel IA - 32 architecture support two levels of page tables.

Advantages

1. It is compact and support sparse address space.
2. It easily manage the memory.
3. It reduce table fragmentation.

Disadvantages

1. Overhead due to one more page table.
2. On TLB miss, two loads from memory will be required to get the right address translation information from the page table.

2) Hashed page tables

- Hashed page table support address space of 32 bits and more. Hash value is used as virtual page number.
- Hash table contains a link list of elements. Each elements consists of following fields :
  1. Virtual page number
  2. Mapped page frame value.
  3. Pointer to the next element.
- Hash table improves search speed.
- Fig. Q.15.2 shows block diagram of hashed page table.

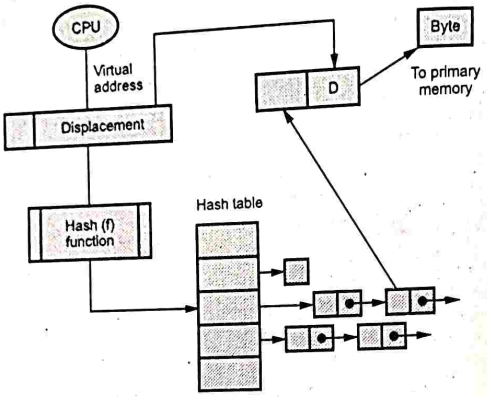


Fig. Q.15.2 Hashed page table

Working

1. Virtual page number is taken from virtual address.
2. Virtual page number is hashed into the hash table.
3. Virtual page number is compared with the first element of linked list.
4. Both the value is matched, that value is (i.e. page frame) used for calculating physical address.
5. If the both value is not matched, the entire linked list is searched for a matching.

- Clustered page table is same as hashed page table but only difference is that each entry in the hash table refers to several pages.

3) Inverted page table

- It uses a page table that contains an entry for each physical frame. This ensure that the page table occupies a fixed fraction of memory. The size is proportional to the physical memory.
- Page table overhead increases with address space size. Page table get too big to fit in memory.
- Inverted page table has one entry for each real page of memory. Lookup time is increased because it requires a search on the inverted table.

- Logical address is as follows

Process id	Page number	Offset

- Inverted page table is used by Itanium, Power PC and Ultra SPARC.

Q.16 Explain with the help of a neat diagram how TLB can be used to improve effective access time ? ☞ [SPPU : Dec.-17, End Sem, Marks 10]

Ans. : • Set of registers are used for implementing page table. Registers are suitable only for small page table. If page table size increases then special purpose hardware cache is used for page table.

- Special cache memory called a Translation Lookaside Buffer (TLB) is used with the address translation hardware. The full page table is kept in primary memory.
- TLB is very effective in improving the performance of page frame access. The cache buffer is implemented in a technology which is faster than the primary memory technology.
- Fig. Q.16.1 shows paging with TLB.

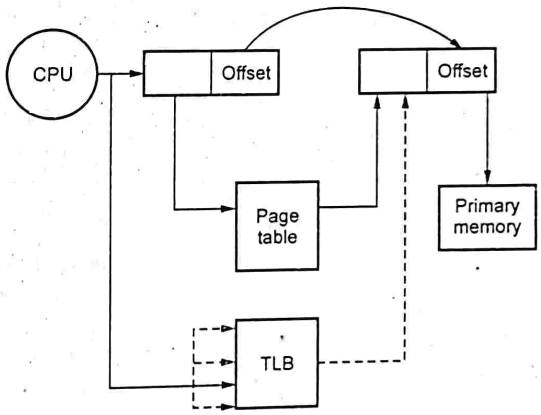


Fig. Q.16.1 Paging with TLB

- The TLB contents may be controlled by the operating system or by hardware, depending on the architecture.

- When a page is first translated to a page frame, the map is read from primary memory into the TLB. The TLB entry contains the page number, page frame's physical address etc.
- The page mapping mechanism first tries to find page number in the TLB. If the TLB contains the page number, then it is called as **page hit** or **TLB hit.**
- If the TLB does not contain an entry for page p, then it called as page miss or TLB miss. In case of TLB miss, the operating system locates the page table entry in the primary memory, which increases execution time.
- The TLB is associative, high-speed memory. A translation buffer is used to store a few of the translation table entries. It is very fast, but only for a small number of entries. On each memory reference.
  1. First ask TLB if it knows about the page. If so, the reference proceeds fast.
  2. If TLB has no information for page, must go through page and segment table to get information. Reference takes a long time, but gives the info for this page to TLB so it will know it for next reference.

Q.17 List advantages and disadvantages of Paging.

Ans. : Advantages of Paging

1. Paging eliminates fragmentation.
2. Support higher degree of multiprogramming.
3. Paging increases memory and processor utilization.
4. Compaction overhead required for the relocatable partition scheme is also eliminated.

Disadvantages of Paging

1. Page address mapping hardware usually increases the cost of the computer.
2. Memory must be used to store the various tables like page table, memory map table etc.
3. Some memory will still be unused if the number of available block is not sufficient for the address spaces of the jobs to be run.

4.5 : Segmentation

Q.18 Explain segmentation with suitable diagram.

☞ [SPPU : Dec.-18, End Sem, Marks 6]

OR Describe the address translation mechanism in segmentation with suitable diagram. ☞ [SPPU : May-19, End Sem, Marks 6]

Ans. : • In segmentation, a program's data and instructions are divided into blocks called segments. A segment is a logical entity in a program.

- Logical view : A process consists of a set of segments.
- Physical view : It consists of non-adjacent areas of memory allocated to segments.
- All segment size may be equal or may not be equal. Segmentation support user view of memory.
- Collection of segment is called as logical address space. Each segment is identified by its name.
- Processor generates logical addresses. These addresses consists of a segment number and an offset into the segment. Segment number is used as an index to segment page table.
- Segment names are normally symbolic names.
- Operating system maintains a segment table for each process. It is usually stored in main memory as a segment that is not to be loaded as long as the process can run.
- Each entry in the segment table has a segment base and a segment limit. The base field contains the segment relocation register for the target segment. The segment limit field contains the length of the segment.
- Fig. Q.18.1 shows an address translation in segmentation.
- Segment table contains the physical address of the start of the segment. Then add the offset to the base and generate the physical address.

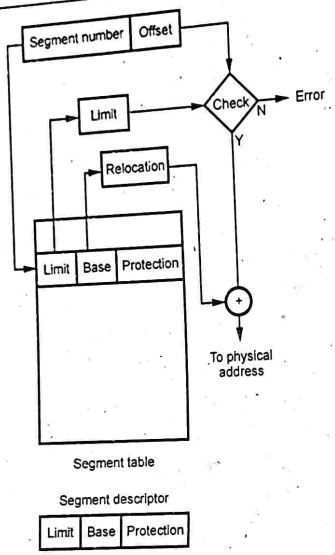


Fig. Q.18.1 Address translation in segmentation

- If the required reference is not found in one of the segment registers, then error is generated. At the same time, operating system does lookup in segment table and loads new segment descriptor into the register.

Q.19 Compare segmentation and paging.

Ans. :

Sr. No.	Segmentation	Paging
1.	Program is divided into variable size segments.	Program is divided into fixed size pages.
2.	User (or compiler) is responsible for dividing the program into segments.	Division into pages is performed by the operating system.

3.	Segmentation is slower than paging.	Paging is faster than segmentation.
4.	Segmentation is visible to the user.	Paging is invisible to the user.
5.	Segmentation eliminates internal fragmentation.	Paging suffers from internal fragmentation.
6.	Segmentation suffers from external fragmentation.	There is no external fragmentation.
7.	Processor uses page number, offset to calculate absolute address.	Processor uses segment number, offset to calculate absolute address.
8.	OS maintain a list of free holes in main memory.	OS must maintain a free frame list.

Q.20 Explain the address translation mechanism in paging and segmentation with proper example. ☞ [SPPU : May-18, End Sem, Marks 8]

Ans. : Refer Q.14 and Q.18.

Q.21 Consider the following segment table :

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses.
i) 0, 430 ii) 1, 10 iii) 2, 80 iv) 3, 400 v) 4, 90

☞ [SPPU : May-16, End sem, Marks 8]

Ans. : Physical Address = Offset + Segment Base

i) 0, 430

Offset = 430

Segment length = 600

Segment = 0

Offset 430 is less than segment length 600 and starting location of segment 0 is start from 219.

Physical address = Offset + Segment Base = 430 + 219 = 649

ii) **1, 10**

Offset = 10

Segment length = 14

Segment = 0

Offset 10 is less than segment length 14 and starting location of segment 1 is start from 2300.

Physical address = Offset + Segment Base = 10 + 2300 = 2310

iii) **2, 80**

Offset = 80

Segment length = 100

Segment = 2

Offset 80 is less than segment length 100 and starting location of segment 2 is start from 90.

Physical address = Offset + Segment Base = 80 + 90

Physical address = 170

iv) **3, 400**

Offset = 400

Segment length = 580

Segment = 3

Offset 400 is less than segment length 580 and starting location of segment 3 is start from 1327.

Physical address = Offset + Segment Base = 400 + 1327 = 1727

v) **4, 90**

Offset = 90

Segment length = 96

Segment = 4

Offset 90 is less than segment length 96 and starting location of segment 4 is start from 1952.

Physical address = Offset + Segment Base = 90 + 1952 = 2042

4.6 : Virtual Memory

Q.22 Explain the concept of virtual memory with suitable diagram.

☞ [SPPU : Dec.-18, End Sem, Marks 8]

Ans. : • Virtual memory is a method of using hard disk space to provide extra memory. It simulates additional main memory. In Windows operating system, the amount of virtual memory available equals the amount of free main memory plus the amount of disk space allocated to the swap file.

- A swap file is an area of your hard disk that is set aside for virtual memory. Swap files can be either temporary or permanent.
- Virtual memory is stored in the secondary storage device. It helps to extend additional memory capacity and work with primary memory to load applications. The virtual memory will reduce the cost of expanding the capacity of physical memory.
- Each process address space is partitioned into parts that can be loaded into primary memory when they are needed and written back to secondary storage otherwise. Address space partitions have been used for the code, data and stack identified by the compiler and relocating hardware.
- The portion of the process that is actually in main memory at any time is defined to be the resident set of the process. The logical addressable space is referred to as virtual memory.
- The virtual address space is much larger than the physical primary memory in a computer system. The virtual memory works with the help of secondary storage device and its speed is low compared to the physical storage location.

- Virtual memory uses two types of addresses : Virtual address and physical address
- Fig. Q.22.1 shows concept of virtual memory.

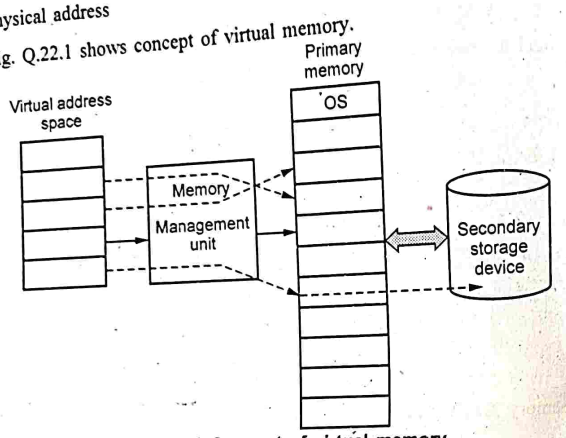


Fig. Q.22.1 Concept of virtual memory

- Virtual address is referred by process and physical address is actual address of the main memory.
- Whenever a process accesses a virtual address, the system must translate it to a physical address. Virtual memory system uses special purpose hardware. This hardware is called as memory management unit.
- Most virtual memory system use a technique called paging. The virtual address space is divided into fixed size units called pages.
- The corresponding units in the physical memory are called page frames. The pages and page frames are normally the same size. The area on a hard disk that stores page frames is usually called the paging file or the swap file.
- When the system is ready to run a process, the system loads the process code and data from secondary storage into primary memory. Only a small portion of these needs to be in primary memory at once for the process to execute. The success of implementing virtual memory system

is how to map virtual addresses to physical addresses. Because processes reference only virtual addresses.

4.7 : Demand Paging

Q.23 Elaborate the concept of demand paging with appropriate diagram. ☞ [SPPU : May-19, End Sem, Marks 8]

Ans. : • In demand paging, a page is loaded into primary memory only when the process references it. It is the simplest fetch policy implemented in virtual memory.

- Demand paging guarantees that the system brings into primary memory only those pages that processes actually need. It involves interaction between hardware and software component of the virtual memory.
- Paging with swapping method will become as demand paging concept. Process's logical address space is stored on the secondary storage device.
- Demand paging requires that the page map table for each process keep track of each page as it is loaded or removed from primary memory.
- Fig. Q.23.1 shows the concept of demand paging.

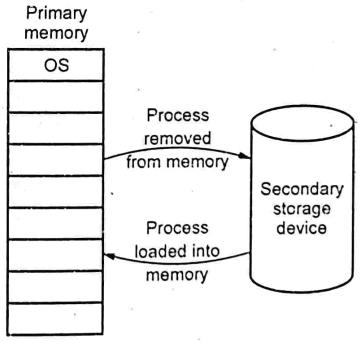


Fig. Q.23.1 Demand paging

- Working of demand paging
  1. Program attempts to find a page.
  2. If the page is located in the primary memory, then the program executes.
  3. If the page cannot be found in the primary memory, then the page fault occurs.
  4. Memory reference is checked for required page to determine it is a valid reference to a location on a secondary storage, if it is then the page will be required page. If not, the process is terminated.
  5. Schedule disk operation to read the required page into primary memory.
  6. Restart the instruction that was interrupted by the OS trap.
- Demand paging takes support from hardware to distinguish the pages. The valid-invalid bit method is used. This method gives the idea about page location. Page may be in the primary memory or in the secondary storage.
- If valid bit is set, then required page is stored in the primary memory. When an invalid bit is set, then page is on the secondary storage disk.
- Valid-invalid bit field is attached with page table. The valid-invalid bit field contain a Boolean value to indicate whether the page exists in primary memory. 0 = Not in memory 1 = In memory
- Initially all valid-invalid entries are set to 0. This information is used for page replacement algorithm. Page fault leads to the page replacement operation to load the required pages in primary memory. A failure to find a page in memory is often called a page fault.

Q.24 What are the steps in handling page fault? Explain with suitable diagram. ☞ [SPPU : May-17, End Sem, Marks 10]

Ans. : Steps in Handling a Page Fault

- When page fault occurs, the system performs following steps :

1. Check the process control block table for reference which bit is set i.e. valid bit or invalid bit for memory access.
2. If invalid bit is set, user will terminate the process. If valid bit is set, then the page is loaded into memory.

3. Check for free frame for new **page loading.**
4. Start reading secondary storage device for desired page for allocated frame.
5. Required page is loaded into the memory after completion of read operation. PCB table is updated.
6. System restart the instruction that was interrupted by the error. Page is loaded in memory so process can use it.

- Fig. Q.24.1 shows page fault handling steps in demand paging.

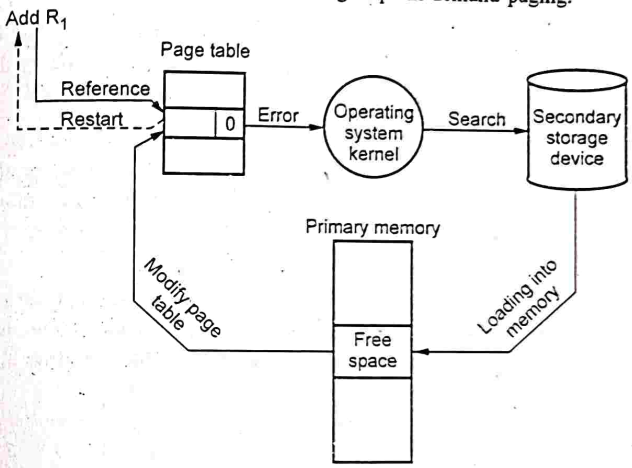


Fig. Q.24.1 Handling of page fault

4.8 : Page Replacement

Q.25 What is page replacement ? List the name of page replacement algorithm.

Ans. : • Whenever a process requests data from a page which is not in the memory then the system uses a page replacement algorithm to swap out an existing page to make space for the new page.

- Page replacement algorithm is used to decide which page will be replaced to allocate memory to the current referenced page.

- There are three page replacement algorithms :
  1. First In First Out (FIFO)
  2. Least Recently Used First (LRU)
  3. Optimal Page Replacement
- In FIFO algorithm the system keeps track of the time when each page gets into the memory. For page replacement, the page which entered first into the memory is selected for replacement. This technique suffers from Beladay's Anomaly.
- LRU page replacement algorithm associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- Optimal page replacement looks in future requests. Optimal page replacement algorithm replaces the page that will not be used for the longest period of time. Unfortunately, there is no way to determine which page will be last, so in practice this algorithm cannot be use.

Q.26 For the following reference string.
1,2,3,4,2,1,5,6,2,1,2,3,3,6. Count the number of page faults that occur with 3 frames using FIFO and LRU page replacement methods. Discuss the result. ☞ [SPPU : De.c-17, End Sem, Marks 8]

Ans. : FIFO :

	1	2	3	4	2	1	5	6	2	1	2	3	3	6
1	1	1	1	4	4	4	4	6	6	6	6	3	3	3
2	-	2	2	2	2	1	1	1	2	2	2	2	2	6
3	-	-	3	3	3	3	5	5	5	1	1	1	1	1
Page Fault	PF	PF	PF	PF		PF	PF	PF	PF	PF		PF		PF

Total number of page fault = 11

LRU :

	1	2	3	4	2	1	5	6	2	1	2	3	3	6
1	1	1	1	4	4	4	5	5	5	1	1	1	1	6
2		2	2	2	2	2	2	6	6	6	6	3	3	3
3			3	3	3	1	1	1	2	2	2	2	2	2
Page Fault	PF	PF	PF	PF		PF	PF	PF	PF	PF		PF		PF

Total number of page fault = 11

Q.27 For the following reference string.
5,6,7,8,5,6, 9,5,6,7,8,9,6,7,4,9,8
Count the number of page fault occur with 3 frames using FIFO, optimal and LRU page replacement methods. Discuss the result. ☞ [SPPU : May-18, End sem, Marks 12]

Ans. : FIFO :

	5	6	7	8	5	6	9	5	6	7	8	9	6	7	4	9	8
1	5	5	5	8	8	8	9	9	9	9	9	9	6	6	6	6	8
2	-	6	6	6	5	5	5	5	5	7	7	7	7	7	4	4	4
3	-	-	7	7	7	6	6	6	6	6	8	8	8	8	8	9	9
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF		PF		PF	PF	PF

Total number of page fault = 13

LRU :

	5	6	7	8	5	6	9	5	6	7	8	9	6	7	4	9	8
1	5	5	5	8	8	8	9	9	9	7	7	7	6	6	6	9	9
2	-	6	6	6	5	5	5	5	5	5	8	8	8	7	7	7	8
3	-	-	7	7	7	6	6	6	6	6	6	9	9	9	4	4	4
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF	PF	PF

Total number of page fault = 15

Optimal :

	5	6	7	8	5	6	9	5	6	7	8	9	6	7	4	9	8
1	5	5	5	5	5	5	5	5	5	7	8	8	8	8	8	8	8
2	-	6	6	6	6	6	6	6	6	6	6	6	6	7	4	4	4
3	-	-	7	8	8	8	9	9	9	9	9	9	9	9	9	9	9
Page fault	PF	PF	PF	PF			PF			PF	PF			PF	PF		

Total number of page fault = 9

Q.28 For the following reference string. 6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 6, 3, 2, 1, 2 Count the number of page faults that occur with 3 frames using FIFO, Optimal and LRU page replacement methods. Discuss the result. ☞ [SPPU : Dec-18, 19, End Sem, Marks 12]

Ans. : FIFO (Page frame = 3)

	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	2	2	2	2	4	4	4	6	6	6	1	1
2	-	5	5	5	5	3	3	3	2	2	2	2	2	2	2
3	-	-	1	1	1	1	5	5	5	3	3	3	3	3	3
Page Fault	PF	PF	PF	PF	PF	PF	PF	PF	PF	PF	PF			PF	

Total number of page fault = 12

Optimal (Page frame = 3)

	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	6	6	3	3	3	3	3	3	3	3	1	1
2	-	5	5	5	5	5	5	4	4	4	6	6	6	6	6
3	-	-	1	2	2	2	2	2	2	2	2	2	2	2	2
Page Fault	PF	PF	PF	PF		PF		PF			PF			PF	

Total number of page fault = 8

LRU (page frame = 3)

	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	2	2	2	2	4	4	4	6	6	6	1	1
2	-	5	5	5	5	5	5	5	5	3	3	3	3	3	3
3	-	-	1	1	1	3	3	3	2	2	2	2	2	2	2
Page Fault	PF	PF	PF	PF		PF		PF	PF		PF			PF	

Total number of page fault = 9

Q.29 For the following reference string 0, 1, 3, 6, 2, 4, 5, 2, 5, 0, 3, 1, 2, 5, 4, 1, 0 count the number of page faults that occur with 3 frames using FIFO, Optimal and LRU page replacement methods. Discuss the results. ☞ [SPPU : May-19, End Sem, Marks 12]

Ans. : FIFO (page frame = 3)

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
1	0	0	0	6	6	6	5	5	5	5	5	1	1	1	4	4	4
2	-	1	1	1	2	2	2	2	2	0	0	0	2	2	2	1	1
3	-	-	3	3	3	4	4	4	4	4	3	3	3	5	5	5	0
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF	PF	PF

Total number of page fault = 15

Optimal (Page frame = 3)

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
1	0	0	0	0	0	0	0	0	0	0	3	1	1	1	1	1	1
2	-	1	1	6	2	2	2	2	2	2	2	2	2	2	4	4	4
3	-	-	3	3	3	4	5	5	5	5	5	5	5	5	5	5	0
Page fault	PF	PF	PF	PF	PF	PF	PF				PF	PF			PF		PF

Total number of page fault = 11

LRU (Page frame = 3)

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
1	0	0	0	6	6	6	5	5	5	5	5	1	1	1	4	4	4
2	-	1	1	1	2	2	2	2	2	2	3	3	3	5	5	5	0
3	-	-	3	3	3	4	4	4	4	0	0	0	2	2	2	1	1
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF	PF	PF

Total number of page fault = 15

4.9 : Allocation of Frames, Thrashing

Q.30 Explain thrashing.

Ans. : • Thrashing occurs when a process does not have "enough" frames allocated to store the pages it uses repeatedly, the page fault rate will be very high.

- A process is thrashing if it is spending more time for paging in/out than executing. Since thrashing leads to low CPU utilization, the operating system's medium or long term scheduler may step in, detect this and increase the degree of multiprogramming.
- Local replacement algorithms can limit the effects of thrashing. If the degree of multiprogramming is increased over a limit, processor utilization falls down considerably because of thrashing. Fig. Q.30.1 shows thrashing.
- The selection of a replacement policy to implement virtual memory plays an important part in the elimination of the potential for thrashing. A policy based on the local mode will tend to limit the effect of thrashing. Replacement policy based on the global mode is more likely to cause thrashing. Since all pages of memory are available to all transactions, a memory-intensive transaction may occupy a large portion of memory, making other transactions susceptible to page faults and resulting in a system that thrashes.

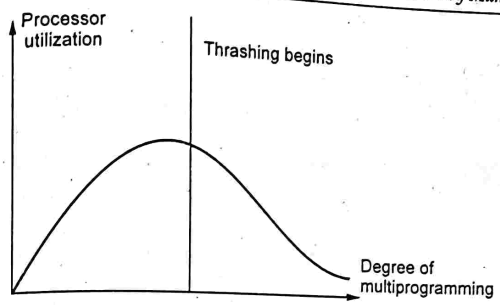


Fig. Q.30.1 Thrashing

- Thrashing is solved by using working set model and page fault frequency.

Q.31 Write short note on locality of reference.

Ans. : • Locality of reference is also known as the principle of locality. Many applications continually reference large amounts of data.

- Locality of reference is observes that an application does not access all of its data at once with equal probability. Instead, it accesses only a small portion of it at any given time.
- There are two basic types of reference locality.
  1. Temporal locality 2. Spatial locality

1. **Temporal locality :** Temporal locality is locality over time. If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.
2. **Spatial locality :** If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.

END...

Unit V

5 Input / Output and File Management

5.1 : I/O Devices, Organization of the I/O Function

Q.1 Enlist the characteristics of block and character devices? Explain each with suitable example. ☞ [SPPU : May-19, End Sem, Marks 4]

Ans. : Characteristics of block devices :

- Driver communicates by sending entire blocks of data.
- Examples for Block Devices : hard disks, USB cameras, Disk-On-Key.
- Block devices are accessed a block at a time.
- Operations supported include read( ), write( ), and seek( ).

Character Devices :

- Character devices are accessed one byte at a time.
- Supported operations include get( ) and put( ).
- Examples for Character Devices : Serial ports, parallel ports, sounds cards.

Q.2 Explain parameters used for differentiating I/O devices.

Ans. :

1. **Data rate** will change according to the input output devices. There may be differences of several orders of magnitude between the data transfer rates. Data transfer rate of keyboard is the lowest among the entire I/O device.
2. **Application :** Different devices have different use in the system.
3. **Complexity of control :** It will change according to the input-output devices. Printer requires a relatively simple control interface. Disk is much more complex interface.

4. **Unit of transfer :** Data may be transferred as a stream of bytes or characters or in larger blocks.
5. **Data representation :** Different data encoding schemes are used for different devices.
6. **Error conditions :** The nature of errors differ widely from one device to another.

Q.3 List and explain in brief I/O performing techniques (at least three). ☞ [SPPU : Dec.-17, End Sem, Marks 12]

OR Discuss at least two techniques for performing I/O. ☞ [SPPU : May-19, End Sem, Marks 8]

Ans. : • I/O performing techniques are DMA, Direct I/O with polling and Interrupt driven I/O.

1. DMA :

- Direct Memory Access (DMA) is an I/O technique that allows a control unit to directly access main memory. This means that once reading or writing has begun the remainder of the data can be transferred to and from memory without processor (CPU) intervention.
- Fig. Q.3.1 shows Operation of the DMA.

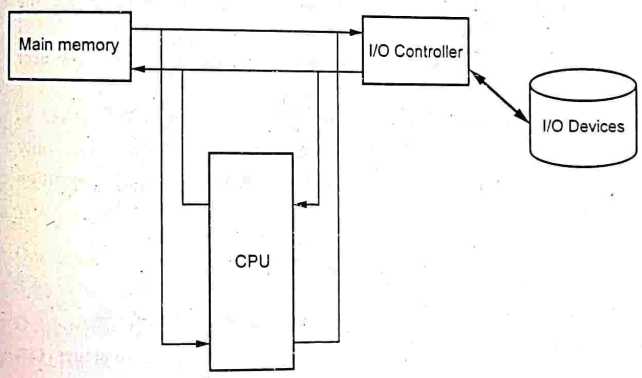


Fig. Q.3.1

- Without DMA, the processor is responsible for the physical movement of data between main memory and a device.
- A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called Direct Memory Access.
- DMA controller is programmed by CPU and its registers are set to perform required operations. It issues a command to a DMA controller for reading data from the secondary storage disk and stores it in the internal buffer. It verifies the checksum. If disk controller buffer contains valid data then DMA operation starts.

2. Direct I/O with polling : CPU is responsible for data transfer. It transfers data between primary memory and device controller. Following are the steps for polling :

a) Application process issues a read operation.

b) Device driver check the status of device. If device is busy, the driver waits for it to become idle.

c) Driver stores an input command into the controller command register. It starts the device.

d) Driver continuously read the status register while waiting for the device to complete its operations.

e) Driver copies the content of the controller data register into the user process space.

3. Interrupt driven I/O : When interrupt I/O is used, the CPU is free to ignore the I/O module until the interrupt signal a received. The only added overhead is processing a single interrupt when the I/O operation completes.

5.2 : I/O Buffering

Q.4 What is I/O buffering? Explain input and output buffering.

Ans. : • Buffering is the technique by which the device manager can keep slower I/O devices busy during times when a process is not requiring I/O operations. Input/output buffering is a mechanism that

improves the throughput of input and output operations. It is implemented directly in hardware and the corresponding drivers.

- Input buffering is the technique of having the input device read information into the primary memory before the process requests it.
- Output buffering is the technique of saving information in memory and then writing it to the device while the process continues execution.

Q.5 Explain in brief different I/O buffering techniques.

☞ [SPPU : Dec.-17, End Sem, Marks 4]

OR Explain I/O buffering mechanism.

☞ [SPPU : May-18, Dec.-18,19, End Sem, Marks 4]

Ans. : Types of I/O buffering schemes are

1. Single buffering
2. Double buffering
3. Circular buffering
4. No buffering

1. Single Buffering

- Operating system assigns a buffer in the system portion of main memory. Fig. Q.5.1 shows single buffering.

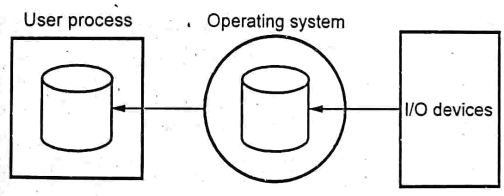
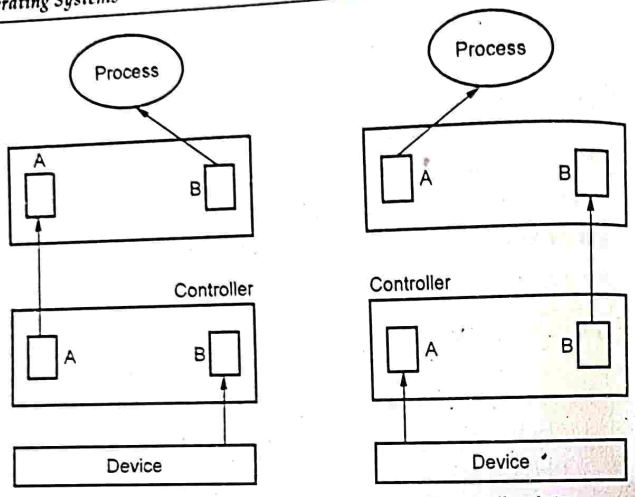


Fig. Q.5.1

2. Double Buffering

- There are two buffers in the system.
- One buffer is for the driver or controller to store data while waiting for it to be retrieved by higher level of the hierarchy.
- Other buffer is to store data from the lower level module.
- Double buffering is also called buffer swapping.
- Fig. Q.5.2 shows the hardware and software double buffering for bytes.



(a) Reading into driver buffer A

(b) Reading into driver buffer B

Fig. Q.5.2 Double buffering in the driver

Circular Buffer

- When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer.
- In this, the producer cannot pass the consumer because it would overwrite buffers before they had been consumed.
- The producer can only fill up to buffer $j-1$ while data in buffer j is waiting to be consumed.
- Fig. Q.5.3 shows circular buffering.

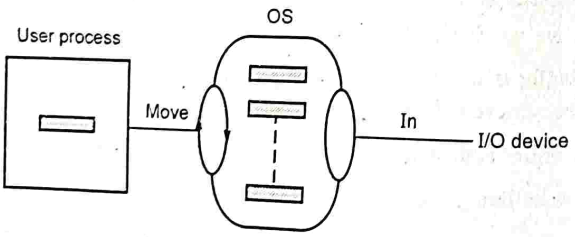


Fig. Q.5.3 Circular buffering

No Buffering : there is a buffer between user process, operating system and I/O device.

5.3 : Disk Scheduling

Q.6 Define following terms : i) Seek time ii) Rotational latency

☞ [SPPU : Dec.-17, End Sem, Marks 4]

Ans. :

1. **Seek time :** The time it takes to position the head at the track is known as **seek time.**

   Seek time = Number of track traversed × Disk drive constant + Startup time

   Seek time do not apply to device with fixed read/write heads.

2. **Rotational latency :** The time it takes for the beginning of the sector to reach the head is known as rotational delay. It is known as search time.

Q.7 Explain SSTF and LOOK disk scheduling algorithm.

Ans. : Shortest Seek Time First

- SSTF select the next request at the one requiring the minimum seek time from the current position.
- In Shortest-Seek-Time-First (SSTF) scheduling priority is given to those processes which need the shortest seek, even if these requests are not the first ones in the queue. It means that all requests nearer to the current head positions are serviced together before moving head to distant tracks.
- Select the disk I/O request that requires the least movement of the disk arm from its current position and always choose the minimum seek time.

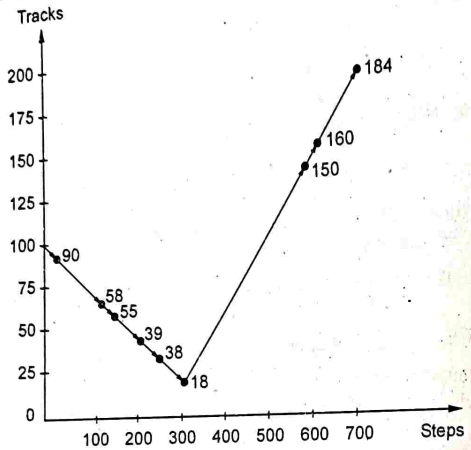


Fig. Q.7.1 SSTF

Next track accessed	Number of track traversed
90	10
58	32
55	03
39	16
38	01
18	20
150	132
160	10
184	24

$$\text{Average seek length} = \frac{10+32+3+16+1+20+132+10+24}{9} = \frac{248}{9}$$

$$= 27.55$$

- The only tricky part is if there are two jobs with the same distance. In this case, some kind of tie-breaking needs to be employed to pick one. For instance, you could just use a random number to effectively flip a coin and pick one.
- SSTF does not ensure fairness and can lead to indefinite postponement because its seek pattern tends to be highly localized.
- Under heavy load, SSTF can prevent distant requests from ever being serviced. This phenomenon is known as starvation. SSTF scheduling is essentially a form of shortest job first scheduling. SSTF scheduling algorithm are not very popular because of two reasons.

  1. Starvation possibly exists. 2. It increases higher overheads.

Advantages :

1. Throughput is better than FCFS.
2. SSTF minimize the response time.
3. Less number of head movements.

Disadvantages :

1. One major issue is STARVATION.
2. Localize under heavy load.

LOOK Scheduling

- Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling, reverse direction and repeat. This algorithm is similar to SCAN, but unlike SCAN, the head does not unnecessarily travel to the innermost and outermost track on each circuit.
- Fig. Q.7.2 shows the look scheduling algorithm. For example, the disk request queue contains a set of references for blocks on tracks 76, 124, 17, 269, 201, 29, 137 and 12.

Current head position = 76

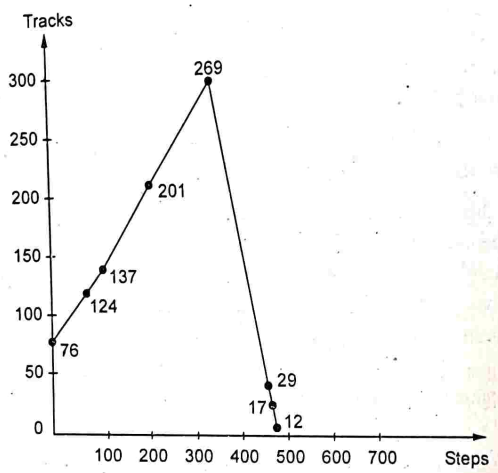


Fig. Q.7.2 LOOK scheduling

Next track accessed	Number of track traversed
124	48
137	13
201	64
269	68
29	240
17	12
12	5

$$\text{Average seek length} = \frac{48+13+64+68+240+12+5}{10} = \frac{450}{7} = 64.28$$

Q.8 Briefly describe SCAN disk scheduling algorithm.

Ans. : • SCAN is also called elevator algorithm.

- The next request scheduled is closest to current request but in one particular direction. All requests in other direction are put at the end of the list. SCAN services tracks in only one direction i.e. either increasing or decreasing track number.
- When SCAN reaches the edge of the disk (or track 0), it reverses direction.
- If any request comes on its way it will be serviced immediately, while request arriving just behind the head will have to wait until disk head moves to the end of the disk, reverses direction and returns before being serviced.

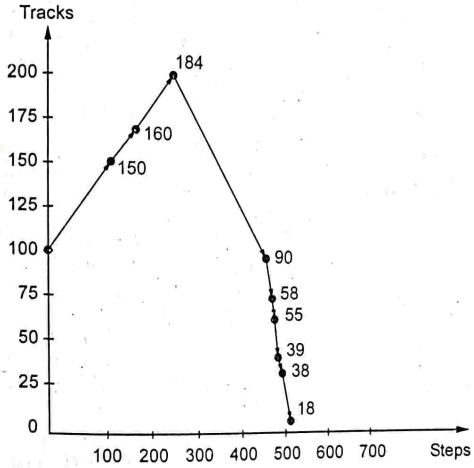


Fig. Q.8.1 SCAN

- SCAN is called elevator because an elevator continues in one direction servicing requests before reversing direction.
- Consider the example in which the requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38, and 184. Starting track

at 100 and total number of track is 200. Head is moving towards increasing track number.

Current head position = 100

Head is moving towards increasing number of track.

Next track accessed	Number of track traversed
150	50
160	10
184	24
90	94
58	32
55	03
39	16
38	01
18	20

$$\text{Average seek length} = \frac{50+10+24+94+32+3+16+01+20}{9} = \frac{250}{9}$$

$$= 27.77$$

Advantages :

1. SCAN eliminates starvation.
2. Throughput similar to SSTF.

Disadvantages :

1. Increase overhead.
2. Needs directional bit.

Q.9 A disk drive has 200 cylinders, numbered 0-199. The drive is currently serving the request at cylinder 53. The queue of pending requests in FIFO order is 98, 183, 37, 122, 14, 124, 65, 67. Starting from the current head position what is the total distance that disk arm moves to satisfy all the pending requests for the following disk scheduling algorithms. i) FCFS ii) SCAN iii) C-LOOK iv) SSTF ☞ [SPPU : Dec.-17, End sem, Marks 12]

Ans. : i) FCFS :

Head movement = 53 → 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67

Total head movement = 640

ii) SCAN :

Head movement = 53 → 37 → 14 → 0 → 65 → 67 → 98 → 122 → 124 → 183

Total head movement = 208

iii) C-LOOK :

Head movement = 53 → 65 → 67 → 98 → 122 → 124 → 183 → 14 → 37

Total head movement = 236

iv) SSTF :

Head movement = 53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183

Total head movement = 236

Q.10 For the given sequence of disk request, determine the total distance travelled by disk head in satisfying the entire request for FCFS, C-SCAN and SSTF algorithms. Initial head position is 120 and total number of cylinders in the disk is 200.

120, 130, 180, 150, 25, 10, 105, 90 ☞ [SPPU : May-17, End sem, Marks 12]

Ans. : i) FCFS :

Head movement = 120 → 130 → 180 → 150 → 25 → 10 → 105 → 90

Total head movement = 340

ii) C-SCAN :

Head movement = 120 → 130 → 150 → 180 → 200 → 1 → 10 → 25 → 90 → 105

Total head movement = 383

iii) SSTF :

Head movement = 120 → 130 → 150 → 180 → 105 → 90 → 25 → 10

Total head movement = 230

Q.11 Is disk scheduling, other than FCFS useful in a single user environment. Explain your answer. ☞ [SPPU : Dec-16, End Sem, Marks 6]

Ans. : • Yes, disk scheduling other than FCFS scheduling can be useful in a single-user environment.

- SSTF is useful when the data sets are near each other. In the elevator scheduling, we saw that the effects are optimal because it is more efficient than FCFS but unlike SSTF, there are no starvation.
- In a single-user environment, the I/O queue usually is empty. Requests generally arrive from a single process for one block or for a sequence of consecutive blocks.
- In these cases, FCFS is an economical method of disk scheduling. But LOOK is nearly as easy to program and will give much better performance when multiple processes are performing concurrent I/O, such as when a Web browser retrieves data in the background while the operating system is paging and another application is active in the foreground.

Q.12 A disk drive has 200 cylinders, numbered 0 - 199. The drive is currently serving the request at cylinder 63. The queue of pending requests in FIFO order is 27, 129, 110, 186, 147, 41, 10, 64, 120. Starting from the current head position what is the total distance that disk arm moves to satisfy all the pending requests for the following disk scheduling algorithms. i) FCFS ii) C-SCAN iii) C-LOOK iv) SSTF ☞ [SPPU : May-18, Dec-18, 19, End Sem, Marks 12]

Ans. : Currently serving the request at cylinder 63.

FCFS		C-SCAN		C-LOOK		SSTF	
Nest Request	Difference in head movement	Nest Request	Difference in head movement	Nest Request	Difference in head movement	Nest Request	Difference in head movement
27	36	64	1	64	1	64	1
129	102	110	46	110	46	41	23
110	19	120	10	120	10	27	14
186	76	129	9	129	9	10	17

147	39	147	18	147	18	110	100
41	106	186	39	186	39	120	10
10	31	0	186	10	176	129	9
64	54	10	10	27	17	147	18
120	56	27	17	41	13	186	39
		41	14				

Total head movement in FCFS = 519

Total head movement in C-SCAN = 349

Total head movement in C- LOOK = 329

Total head movement in SSTF = 231

Q.13 Suppose that a disk drive has 400 cylinders, numbered 0 to 399. The drive is currently serving a request at cylinder 160. The queue of pending requests, in FIFO order, is 370, 30, 390, 130, 310, 170, 340, 180. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms ?

i) FCFS ii) SSTF iii) SCAN iv) LOOK

(Assume that the head is initially moving towards track number 399). ☞ [SPPU : May-19, End Sem, Marks 12]

Ans. : Disk scheduling algorithm : Starting head position at 160

FCFS		SSTF		SCAN		LOOK	
Next request	Difference in head movement	Next request	Difference in head movement	Next request	Difference in head movement	Next request	Difference in head movement
370	210	170	10	170	10	170	10
30	340	180	10	180	10	180	10
390	360	130	50	310	130	310	130
130	260	30	100	340	30	340	30
310	180	310	280	370	30	370	30

170	240	340	30	390	20	390	20
340	170	370	30	399	9	130	260
180	160	390	20	130	269	30	100
Total	1920	Total	530	30	100	Total	590
				Total	608		

5.4 : Overview of File Management

Q.14 Discuss the following with respect to file system : file attributes.

Ans. : File Attributes

- One of the characteristics of file is a set of file attributes that give the operating system more information about the file and how it is intended to be used. For human users convenience bane is given to the file. File attributes are varies from system to system. File attributes are as follows :

1. Name 2. Identifier 3. Type 4. Location/place
5. Size 6. Protection 7. Date and time

- **File name** is in human readable. User can perform any operation on file using its name.
- When file is created by user, operating system assigns unique identification number to each file. OS uses this **identifier** for its operation.
- **Type** information is required for systems that support different types of files.
- **Location/place** information is pointer to a device and actual location of files on the device. The information needed to locate the file on disk is kept in memory.
- **Size** is measured in bits or bytes. It gives idea about current size of the file.
- **Protection :** This information is stored on the per-process table so the operating system can allow or deny subsequent requests. Attributes like protection, password, creator and owner provides protection to a file.

- **Date and time :** This information is related to creation and modification of files. Protection, security and monitoring purposes this data is used by operating system.

Q.15 Discuss the following with respect to file system : file operation

Ans. : • File operations are simply those things that user can perform on a file. For example, user can create a file, save a file, open a file, read a file and modify a file. There are many different types of file operations supported by operating system.

- Operating system can provides system call for performing various operations on the file. Six basic file operations are creating, write, read, delete, repositioning and truncating.
- Following are the some list of file operation :

1. Create 2. Write
3. Close 4. Read
5. Delete 6. Repositioning

- All these operations require that the directory structure be first searched for the target file.
- **File creation :** Any time user can create a file. File is also created without data. The steps for creating a file.

a. Space : File system must provide a space for the file.
b. New file entry is made in the directory.

- **Write :** To perform write operation on the file, file name and data is required. It updates a file by adding new data and changes some content of the file. For writing into a file, operating system searches the directory for particular file. Write pointer is kept at the location where the writing starts. Write pointer position is at middle of the file or end of the file. Write pointer is updated when writing to file is completed.
- **Close :** Process is not used a file after closing. Process can not perform any operation on file. Operating system remove it's entry from open file table.
- **Read :** A process reads all or some portion of the data in a file. Read system call is used for reading a file. For reading a file, name and

position of the memory is specified. Read pointer is kept into the file for next read operation. Read pointer is updated after completion of the read system call. Information about read pointer is stored in *per process current file position pointer.*

- **Delete :** Operating system remove the file from file structure. For deleting a file, directory is searched with particular name and file is deleted. Deleting means making space free from memory and disk. Operating system also removes the directory entry. This free space is used by another file.
- **Repositioning :** The directory is searched for the proper entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seeks.
- Other file operations are open, append, seek, get and set attributes and rename.
- **Open :** Before performing any operation on file, it is necessary to open file in required mode.
- **Append :** Append is related to write operation of file. Append only add data to the end of the file.
- **Get and set attributes :** Get attribute is used before performing operation on file. When user request a file for any operation, get attributes is executed by the process. Depending upon the attributes, user can allow or deny the operation. User can set the attributes and it changed after creation of file.
- **Rename :** User can change the name of existing file.

Q.16 What is the objective of file management system ?

Ans. : Objective of file management system

1. Provide I/O support the multiple users.
2. Provide I/O support for a variety of secondary storage device.
3. Support for data management
4. It minimizes the data lost.
5. Optimize the system performance.

Q.17 Draw and explain file system architecture.

Ans. : File System Architecture

- Fig. Q.17.1 shows file system architecture.

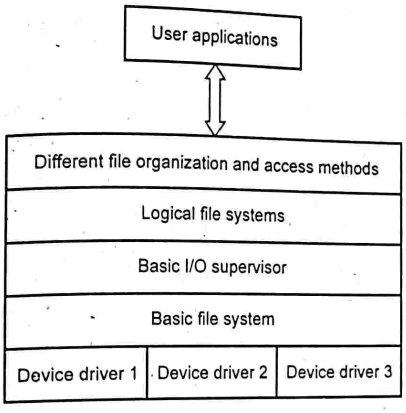


Fig. Q.17.1 File system architecture

Device driver

- Device driver directly communicate with peripheral device and their controllers.
- It is responsible for starting physical I/O operations on the device.

Basic file system

- It uses specific device driver.
- It is primary interface used by system for communicating with outside world.
- It exchange block of data with secondary storage device.
- It is part of the operating system.

Basic I/O Supervisor

- It is responsible for initialize all file I/O and termination.
- It is also part of operating systems.
- It select the device on which file I/O is to be performed.
- It allocates I/O buffer.

5.5 : File Organization and Access

Q.18 Explain different file organization techniques ?

☞ [SPPU : Dec.-16, End Sem, Marks 8]

Ans. : • Different file organization techniques are Sequential Access, direct access, Indexed sequential access.

Sequential Access Method

- Sequential access method is simple method. The information in a file is accessed sequentially one record after another.
- In this method, a process could read all the records in a file in order, starting at the beginning. It can not skip any records and can not read out of order. Fig. Q.18.1 shows sequential access method.

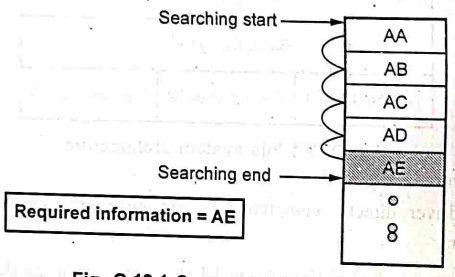


Fig. Q.18.1 Sequential file access

- Batch application uses sequential files. A byte stream file uses sequential access method.
- **Example :** Compiler and editor usually access files in this method. Transaction file is also example of sequential access method.
- Sequential file organization only method easily stored on tape and hard disk. Sequential access is best suited where most of the records in a file are to be processed.

Disadvantages :

- It provides poor performances.
- More efficient search technique is required.

Direct Access Method

- It is also called random access method. This access allows a user to position the read/write mark before reading or writing. Fig. Q.18.2 shows direct access method.

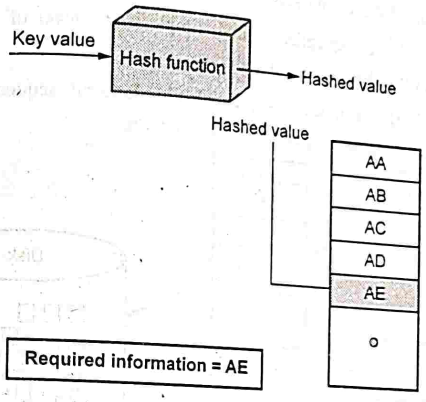


Fig. Q.18.2 Direct access method

- Direct access file method provides, accessing the records directly. Direct access method is based on the hard disk that is a direct access device. It allows random access of any file block.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing. This feature is used by editors. An editors need to randomly access the contents of the file.
- There is no restriction on the order of reading or writing for a direct access file. Operating system support is not required for direct access file.
- At the time of file creation, access method is defined. According to defined access method, file is accessed. Sequential access of a direct access file is possible but direct access of a sequential file is not.

Disadvantages :

1. Poor utilization of input-output device.

2. Consumes CPU time for record address calculation.

Indexed Sequential Access

- It is modified method of direct access method. This method is combination of direct and sequential access method.
- In simple indexed sequential method, simple single level of indexing is used. Sequential file is used as index.
- Fig. Q.18.3 shows indexed sequential file. Indexed sequential access contains key field and a pointer to the main file.

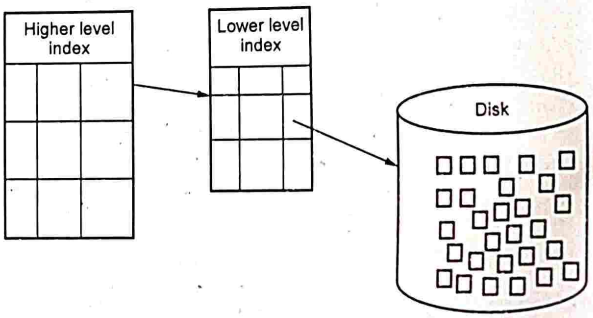


Fig. Q.18.3 Indexed sequential access

- Indexed sequential files are store records in the order that they are written to the disk. Records may be retrieved in sequential order or in random order using an index to represent the record number in the file.
- If file size is large, larger index is required and it contains large number of entries. CPU takes time to search into the index. Higher level index is used to reduce the search time. An entry in the higher level index points to a section of the index. This higher index contains the records. This index is searched to find the section of the disk that may contains a required record.
- File system maintains an overflow file. It adds new records to an overflow file. The record in the main file that immediately precedes the new record in logical sequence is updated to contain pointer to the new record in the overflow file.

- The overflow file is merged with the main file during a batch update. The multiple indexes for the same key field can be set up to increase efficiency.
- The lowest level of index file is treated as a sequential file and a higher level index file is created for that file. Higher index file would contain pointers to lower index files, which would point to the actual data items.

5.6 : File Directories

Q.19 Explain various operation performed on directory.

Ans. : Operation on directory :

1. **File searching :** Directory structure is searched for particular file entry. File uses symbolic names and similar names may indicate a relationship between files.
2. **Create a file :** User creates new files and added to the directory.
3. **Delete a file :** User remove the file after completing its work on files.
4. **Rename a file :** User change the file name if file content is change.
5. **Listing of directory :** Listing of files in the directory for some use. MS-DOS and windows uses "dir" command and Linux/UNIX uses "ls" command for this purposes.

Q.20 Write a short note on : Directory structure.

☞ [SPPU : May-18, Dec-18,19, End Sem, Marks 4]

Ans. : • Directories are basically symbol tables of files. A single flat directory can contain a list of all files in a system. A directory contains information about the files, including attributes, location and ownership.

- All information about files is kept in the directory structure. It is stored on the secondary storage device.
- **Single level directory :** Single level directory is a simple structure and there are no sub-directories. It is also called flat directory structure.
- In single level directory structure, no two files can have the same name.
- **Two level directory :** Two-level directory structure contains master file directory at the top level then user file directory at the second level. Actual user files are at the third level.

- File system maintains a master block for each user. Master block has one entry for each user. User directory address is stored in the master block. Each user has a private directory.
- **Tree structured directory :** In this structure, directory itself is a file. A directory and sub directory contains a set of files. Internal format is same for all directories. Commonly used directory structure is tree structure. Tree has a root directory. All files in disk have a unique path name.

Q.21 Explain acyclic graph tree.

Ans. : • Acyclic graph directory solve the problem of tree structure directory. It allows sharing the directory in between two users. At a time more than one places, shared directory or file will exist in the file system. Fig. 5.8.4 shows acyclic graph directory. Links can be used to construct acyclic graph structure. Acyclic graph is graph with no cycles.

- It is very interesting to note that a shared directory or file is not the same as two copies of the file. When there are two copies of files, each

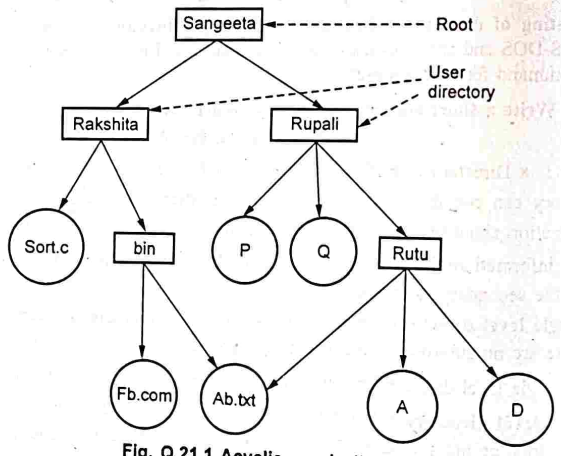


Fig. Q.21.1 Acyclic graph directory

user can view the copy rather than the original, but if one user changes the file content, the changes will not appear in the other's copy.

- Only one original file exists for shared copy. Any changes made by one user are immediately visible to the other user. When user create file in shared directory, it automatically appear in all the shared subdirectories.

5.7 : File Sharing

Q.22 Write a short note on : File sharing.

☞ [SPPU : May-18, Dec-18,19, End Sem, Marks 4]

Ans. : • Multiprogramming and multitasking operating system support the multiple user. These users perform various operations on the files. Files are shared between number of users. File protection, naming and sharing is issue for file sharing.

- Directory structure may allows files to be shared by users. Sharing must be done through a **protection** scheme.
- In single user system, operating system maintains attributes for all files and directory. According to the attributes, operating system allows to operate the files for users. But on the multi-user system, more attributes are needed. An attributes like owner, group and other user are required. These are the access permission for files and directory.
- Access permissions are of three types : Read, Write and execute.
  1. An "r" indicates read permissions for a file or directory.
  2. A "w" indicates write permissions.
  3. An "x" indicates execution permissions if it is a file and search permissions if it is a directory.
  4. Owner is the file owner who created a file. Owner have all rights (read, write and execute).
  5. The groups of other user have some limited access permission. Mostly read and execute permission.
  6. Last categories of user have only executed permission.
- **Example :** In UNIX operating system, file owner can perform all operation on a file. Group member can execute one subset of those

operations and other than the group user can execute another subset of operations.

- When a user requests an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file. Likewise, the group IDs can be compared.

5.8 : Record Blocking

Q.23 List and briefly define three blocking methods.

Ans. : • File is said to employ blocking of records if a physical record contains more than one logical record. Records are the term used to refer to a number of related values or items. Each value or item is stored in a field of a specific data type. Records may be of either fixed or variable lengths.

- Blocks are of fixed length. Disks are divided into concentric circular tracks on each disk surface. A track is divided into blocks. Whole blocks are transferred between disk and main memory for processing.
- Blocking is refers to storing a number of records in one block on the disk. The blocking factor refers to the number of records per block. There may be empty space in a block if an integral number of records do not fit in one block.
- Spanned records are referring to records that exceed the size of one or more blocks and hence span a number of blocks.
- Record blocking are of three types :
  1. Fixed blocking.
  2. Variable length spanned blocking.
  3. Variable length unspanned blocking

1. Fixed blocking

- Fixed blocking uses fixed length records. Fig. Q.23.1 shows fixed blocking.

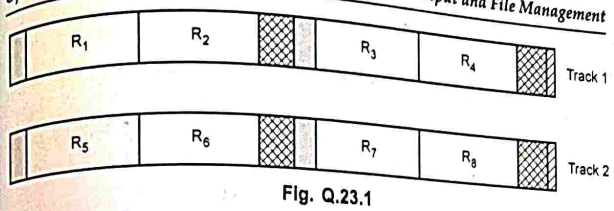


Fig. Q.23.1

- After allocating data, it creates unused space at the end of each block. It is called internal fragmentation. Internal fragmentation is stored in a block.
- Fixed blocking is common for sequential files with fixed-length records.

2. Variable length spanned blocking

- In this blocking method, record sizes are not same. It uses variable length records. These records are packed into blocks without wastage of space.
- Some record may consume two blocks. In this situation, pointer is passed from one block to another block.
- It is efficient in length and efficiency of storage. It doesn't limit record size, but it is more complicated to implement. So, the files are more difficult to update.

3. Variable length unspanned blocking

- It also uses variable length records. But records are not spanned between the blocks. It creates larger waste space as compared to other method.
- If next record is larger that the spanned size, then it creates unused space.

5.9 : Secondary Storage Management

Q.24 Explain contiguous file allocation.

Ans. : • When user creates a file, system allocates a set of contiguous blocks on disk. This is a pre-allocation method that uses portion of variable size. Contiguous allocation is simple method to implement. It

only search free list of correct number of consecutive blocks and mark them as used block.

- Disk address is a linear ordering on the disk. Because of linear ordering, accessing block b + 1 after block b normally requires no head movement. Here head movement is only one track. Fig. Q.24.1 shows contiguous allocation.

File allocation table

File name	Start block number	Block length
File 1	1	3
File 2	18	2
File 4	6	6
File 5	20	1
File 6	25	3

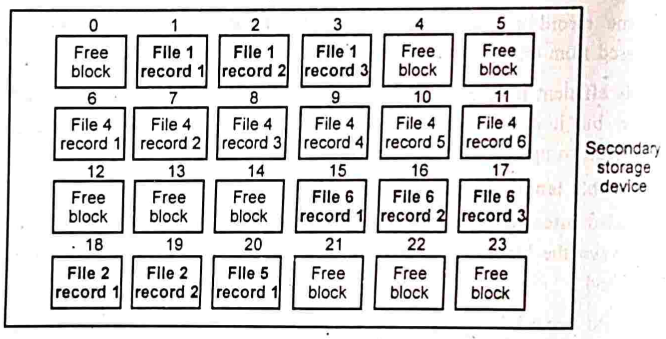


Fig. Q.24.1 Contiguous allocation

- Contiguous allocation of a file is defined by the disk address and the length of the first block.
- If the file size is "n" blocks and starting location is "L", then it occupies blocks L, L+1, L+2, L+3, ..., L+(n–1). The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

- Sequential and random access can be supported by contiguous allocation.
- It is easy to retrieve single block. To improve the I/O performance of sequential processing, multiple blocks can be brought in one at a time. It supports sequential access very well because files entire data is stored in adjacent block. This method also supports random access.
- Contiguous allocation also suffers from **external fragmentation.** Small free disk spaces are created after allocation free block and deleting files. External fragmentation means there will require free space available but that is not contiguous. To solve the problem of external fragmentation, compaction method is used.
- One more problem is that how to calculate the space needed for a file. It is difficult to estimate the required space.

Q.25 Describe indexed allocation with its advantages and disadvantages.

Ans. : • Indexed allocation method solves the problem of both contiguous and linked allocation. It uses concept of index block. Index block stores the entire pointer in one location. But the index block will occupy some space and thus could be considered as an overhead of the method.

- OS keeps a list of free blocks. It allocates an array to hold pointers to all the blocks used by the file. It allocates blocks only on demand. Fig. Q.25.1 shows indexed allocation. (Refer Fig. Q.25.1 on next page)
- In indexed allocation, each file maintains its own index block. It contains an array of disk sector of addresses. For example : The nth entry in the index block points to the nth sector of the file. The directory contains the address of the index block of a file. To read the nth sector of the file, the pointer in the nth index block entry is read to find the desired sector.
- It supports direct access and without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.

File name	Index block number
File 1	10
File 4	24

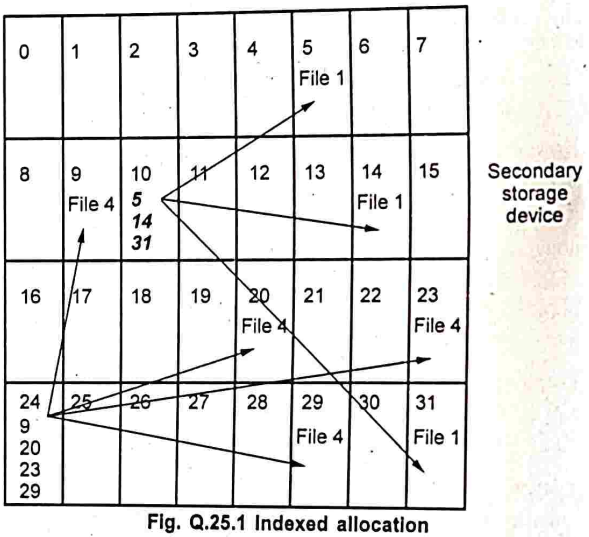


Fig. Q.25.1 Indexed allocation

- Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Advantages :

1. It supports sequential and direct access.
2. No external fragmentation.
3. Faster than other two methods.
4. It supports fixed and variable size blocks.

Disadvantages :

1. Indexed allocation does suffer wasted space.
2. Pointer overhead is generally greater.

Q.26 Compare contiguous, linked and indexed file allocation methods.

Ans. :

Sr. No.	Contiguous	Linked	Indexed
1.	Allocate each file to contiguous blocks on disk	Allocate linked-list of fixed-sized blocks	Allocate fixed-sized blocks for each file
2.	It suffers from external fragmentation.	No external fragmentation	No external fragmentation
3.	Pre-allocation is required	Pre-allocation is possible	Pre-allocation is possible
4.	It support variable size portions.	It support fixed size portions.	It supports fixed and variable size portions.
5.	Simple to calculate random addresses	Cannot calculate random addresses without reading previous blocks	Supports random access
6.	Allocation frequency is once	Allocation frequency is low to high	Allocation frequency is medium
7.	No pointer overhead	Space required for pointer	Pointer overhead is generally greater

Q.27 List and explain any two file allocation mechanisms.

[SPPU : May-19, End Sem, Marks 8]

Ans. : Refer Q.26.

Q.28 Explain advantages and disadvantages of contiguous and linked allocation methods.

Ans. : Advantages of contiguous allocation methods

1. It supports variable size portion.
2. Easy to retrieve single block.
3. Accessing a file is easy.
4. It provides good performance.

Disadvantages of contiguous allocation methods

1. It suffers from external fragmentation.
2. Pre-allocation is required.

Advantages of linked allocation methods

1. There is no external fragmentation.
2. It is never necessary to compact disk space.
3. Pre-allocation is not required.

Disadvantages of linked allocation methods

1. Files are accessed only sequentially.
2. Space required for pointers.
3. Reliability is not good.
4. Can not support direct access.

END...

Unit VI

6 Systems Software and its Importance

6.1 : Study of Various Components of System Software

Q.1 Define system software. List example of system software.

Ans. : Definition : Systems software consists of a set of programs that support the operation of a computer system and help, the programmer, to simplify the programming process and create a environment to run application software efficiently.

- Examples of systems software are : *Text editors, Compilers, Loaders, Linker, Debugger, Assembler and Operating system.*
- Software is of two types : *System software and application software.*

Q.2 What is debugging? Explain classification of debugger.

Ans. : • Debugging means locating and then removing bugs, i.e., faults, in programs. A debugger is a tool that allows user to examine the state of a running program.

- An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs. **Debugging** is a methodical process of finding and reducing the number of bugs, or defects, in a computer program.
- Classification of Debuggers are as follows :

1. **Static debugger :** Static debugger is not depends upon any specific software. User performed debugging task itself.
2. **Dynamic debugging :** Dynamic debugging is either software or hardware debugger. List of dynamic debugger are as follows :
   a. **Breakpoint debugger :** Setting conditional and unconditional breakpoints at various positions in the program.

b. **Kernel debugger :** Debugger with kernel debugging facilities is provided to debug the operating system.

c. **Meta debugger :** Debugger provides debugging facility with meta debugging facility.

d. **Debugger with same process :** Process of debugger and debuggee are same and both share the main memory also. This type of debugger is simple and easy to implement. This debugger is faster in execution.

Q.3 Explain system software in terms of program development environment.

Ans. : Systems Software : Program Development Environment

- **Text editor :** Software that permits the creation and editing of text files (i.e. application programs).
- **Compiler :** Translates programs written in a high level language to object code or machine code.
- **Assembler :** Translates programs written in assembly language to object code or machine code.
- **Static Linker :** Combines and resolves references between object programs and creates the executable code.
- **Debugger (Low-level) :** It is used to debug executable programs and their related object code and source program.

6.2 : Assemblers

Q.4 What is assembler? List assembler function.

Ans. : • The assembler is responsible for translating the assembly language program into machine code. When the source language is essentially a symbolic representation for a numerical machine language, the translator is called an **assembler** and the source language is called an assembly language.

- A pure assembly language is a language in which each statement produces exactly one machine instruction. Fig. Q.4.1 shows the role of assembler.

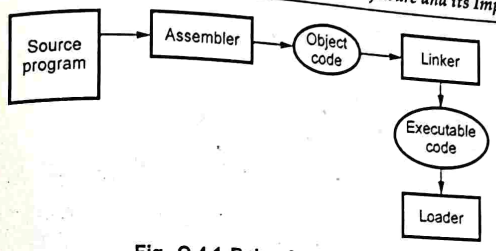


Fig. Q.4.1 Role of assembler

Assembler functions

1. Translate mnemonic opcodes to machine language.
2. Convert symbolic operands to their machine addresses.
3. Build machine instructions in the proper format.
4. Convert data constants into machine representation.
5. Error checking is provided.
6. Changes can be quickly and easily incorporated with a reassembly.
7. Variables are represented by symbolic names, not as memory locations.

Q.5 With syntax and example types of assembly language statements.

☞ [SPPU : April-19, In Sem, Marks 6]

Ans. : It consists of three types of statements :

1. Imperative 2. Declaration 3. Assembler directives

Imperative statement : It specify an action to be performed during the execution of the assembled program. Imperative programs focus on how to solve a problem popular imperative languages such as C and Pascal are based on side effects on memory. Each imperative statement translates into one machine instruction.

Declaration statement : It focus on what the problem is and leave the solution mechanism up to the language implementation. Within the declarative paradigm, most notable are the functional languages such as Lisp. Declarative languages are typically quite abstract and hence can be harder to implement efficiently.

Assembler Directive

Assembler directives instruct the assembler to perform certain actions during the assembly of a program. They can be used to declare variables, create storage space for results, to declare constants. The following assembler directives are used in the program :

1. START - Specify name and starting address for the program.

   START< constant >

2. END - Indicate the end of the source program and specify the first executable instruction in the program.

   END [< operand spec >]

3. BYTE - Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.
4. WORD - Generate one word integer constant.

6.3 : Simple Assembly Scheme

Q.6 Explain steps for design specification of an assembler.

Ans. : • Following steps are used for the design specification of an assembler

1. Find out the information necessary to perform a task.
2. Specify data structure and define format of data structure.
3. To obtain and maintain the information, determine required processing.
4. To perform the task, determine required processing.

Q.7 Explain task of analysis phase and synthesis phase.

Ans. : Analysis phase task

1. Separate the label, mnemonic opcode and operand fields of a statement.
2. If statement contains label then the pair of (symbol, < LC content >) is entered in the symbol table.
3. Validity of the mnemonic opcode is checked through a look-up table by using mnemonics table.
4. Location counter processing is done.

Synthesis phase task

1. Read the machine opcode corresponding to the mnemonic from the mnemonics table.

2. Get an address of a memory operand from the symbol table.
3. Synthesize a machine instruction.

6.4 : Pass Structure of Assembler

Q.8 Describe two pass translation.

Ans. : • Forward reference is handled easily in two pass translation of the assembly language program. Location counter processing is performed in the first pass and symbols defined in the program are entered into the symbol table.

• Fig. Q.8.1 shows two pass structure.

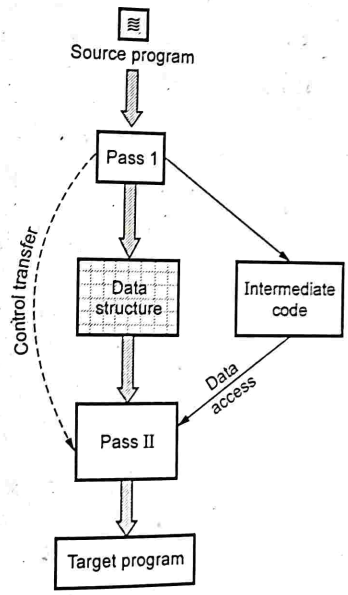


Fig. Q.8.1 Two pass assembly

• The second pass synthesizes the target from using the address information found in the symbol table.

- Functions of the two passes of a simple assembler are as follows.
  1. Assign addresses to all statements in the program.
  2. Save the values assigned to all labels for use in Pass 2.
  3. Perform some processing of assembler directives.

Pass 2 (Assemble instructions and generate object program)

1. Assembler instructions (i.e. translating operation codes and looking up address)
2. Generates data values defined by BYTE, WORD etc.
3. Perform processing of assembler directives not done during Pass I.
4. Write the object program and the assembly listing.

Q.9 Explain in detail Operation code table and symbol table.

Ans. : • Simple assembler uses two major internal data structures.

a) The Operation Code Table (OPTAB)

b) The Symbol Table (SYMTAB)

- OPTAB contains the mnemonic operation and its machine language equivalents. It also contains instruction format and length.
- **OPTAB** is used to look up mnemonic operation codes and translate them to their machine language equivalents. **SYMTAB** is used to store values assigned to lables.
- Location counter (LOCCTR) is also required. LOCCTR is a variable used to help in the assignment of addresses. It is initialized to the beginning address specified in the START statement.
- After processing every source statement, the length of the assembled instruction or data to be generated is added to LOCCTR
- OPTAB must contain the mnemonic operation code and its machine language equivalent. This table also contains information about instruction format and length for complex assembler.
- **In Pass 1,** OPTAB is used to look up and validate operation codes in the source program. **In Pass 2** it is used to translate the operation codes to machine language.

- OPTAB is usually organized as a hash table, with mnemonic operation code as the key. OPTAB is a static table, i.e. entries are not normally added to or deleted from it.
- SYMTAB includes the name and value for each label in the source program, together with flags to indicate error conditions. It also contains information about the data area or instruction labeled.
- **During Pass 1,** labels are entered into SYMTAB as they are encountered in the source program along with their assigned address.
- **During Pass 2,** symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.
- SYMTAB is also organized as a hash table for efficiency of insertion and retrieval. Entries are rarely deleted from this table.
- It is possible for both passes of the assembler to read the original source program as input. Information such as location counter values and error flag statement can be communicated between the two passes. For this reason, Pass 1 usually writes an intermediate file that contains each source statement together with its assigned address, error indications etc. This file is used as the input to Pass 2.

6.5 : Design of Two Pass Assembler

Q.10 Briefly discuss advanced Assembler Directive.

Ans. : 1. ORIGIN (ORG)

- This can be used to indirectly assign values to symbols.
- When this statement is encountered during assembly of a program, the assembler resets its location counter to the specified value.
- The ORG statement will thus affect the values of all labels defined until the next ORG.
- Normally when an ORG without specified value is encountered, the previously saved location counter value is restored.
- Syntax of the ORIGIN directive is :

  ORIGIN *<address spec>*

where <address spec> is an <operand spec> or <constant>.

- This type of statement is useful when the target program does not consist of consecutive memory words.
- No Forward Reference is allowed.
- For example
  a. SYMBOL : 6 bytes
  b. VALUE : 1 word
  c. FLAGS : 2 bytes
  d. LDA VALUE, X

2. EQU

- Most assemblers provide an assembler directive that allows the programmer to define symbols and specify their values, for example. The general form :
  symbol EQU value
- One common use of EQU is to establish symbolic names that can be used for improved readability in place of numeric values.
- Another common use of EQU is in defining *mnemonic names* for *registers*.
  For example :
  A EQU 0
  X EQU 1
  L EQU 2
- These statements cause the symbols A, X, L, ,,, to be entered into SYMBOL with their corresponding values 0, 1, 2, ...

3. LTORG

- LTORG allows placing literals into a pool at some other location in the object program.
- Directive LTORG creates a literal pool that contains all of the literal operands used since the previous LTORG or the beginning of the program.

- Literals placed in a pool by LTORG will not be repeated in the pool at the end of the program.
- The LTORG statement permits a programmer to specify where literals should be placed. By default, assembler places the literals after the END statement.
- Assembler allocates memory to the literals of a literal pool. The pool contains all literals used in the program since the start of the program or since the last LTORG statement.

Q.11 What is forward reference ? How forward reference problem is solved ?

Ans. : • Once declared, identifiers can be used freely. Using an identifier before its declaration is called a forward reference. The use of a symbol at one point in the text which is not defined until some later point gives the wrong result.

- Forward references in programs cannot be resolved in a single pass. So we need more than one pass through the source code.
- General solution to the forward reference problem rests on the observation that all of the referenced symbols should be defined after processing a program even though they may not be defined when they are needed during processing.
- Therefore, if the output of the assembler after processing the input once is discarded, except for the contents of the symbol table, and then the input is processed again, with the symbol table already loaded.
- Forward reference is handled easily in two pass translation of the assembly language program.
- Location counter (LC) processing is performed in the first pass and symbols defined in the program are entered into the symbol table.
- In two pass translation, LC processing and construction of the symbol table is done. **Back patching** is used to solve the problem of forward reference.

- The operand field of an instruction containing a forward reference is left blank initially. The address of the forward reference symbol is put into this field when its definition is encountered.
- For example : **MOVER BREG, ONE**
- In the above statement, ONE is a forward reference. The memory location 101 contains the instruction opcode and address of BREG.
- Table of Incomplete Instructions (TII) is used for inserting the second operand address. TII contains the **(< instruction address >, < symbol >)**
- When assembler reads END statement, the symbol table would contain the addresses of all symbols defined in the source program and table of incomplete instruction would contain information describing all forward references.
- The assembler can now process each entry in TII to complete the concerned instruction.

Q.12 Explain difference between One Pass and Two Pass Assembler.

Ans. :

Sr. No.	One Pass	Two Pass
1.	It performs translation in one pass.	It performs translation in two pass.
2.	It scans entire file only once.	It requires two passes to scan source file.
3.	It generates intermediate code.	It does not generate intermediate code.
4.	Data Structure : Symbol table, literal table, Pool table and table of incomplete instructions.	Data Structure : Symbol table, literal table and Pool table
5.	Perform some processing of assembler directives	Perform processing of assembler directives not done in Pass-1

6.	It is faster than two pass assembler	It is slower than one pass assembler
7.	No object program is written; hence no loader is required.	Loader is required as object code is generated.

6.6 : Introduction to Macro Processors

Q.13 Define macro.

Ans. : Macro is a unit of specification for program generation through expansion. A macro consists of a name, a set of formal parameters and body of code.

Q.14 What are the difference between macro and functions ?

Ans. :

Macro	Function
Macro is Preprocessed	Function is Compiled
Macro does not support type checking	Function support type checking
Program length increases	Program length remain same
There is side effect while using macro.	No side effect while using function
Execution is Faster	Execution is slower
Useful where small source code appears many time	Useful where large source code appears many time
Macro does not Check Compile Errors	Function Checks Compile Errors
Before Compilation macro name is replaced by macro value	During function call, transfer of Control takes place

6.7 : Macro Expansion

Q.15 Explain the lexical expansion of Macro with example.

☞ [SPPU : Aug.-14, Oct-16, In Sem, Marks 6]

Ans. : • Lexical expansion is replacement of a character string by another character string during program generation.

Lexical substitution consists of three types of strings

a) Ordinary string

b) Name of a formal parameter which is preceded by the character '&'

c) Name of a preprocessor variable

- In lexical expansion, only string type 1 is retained without substitution. Other two string types are replaced by the values of the formal parameters or preprocessor variables. Actual parameter string is used as the value of a formal parameter.
- Lexical expansion is to replace occurrences of formal parameters by corresponding actual parameters.

Example

- The following sequence of instructions is used to increment the value in a memory word by a constant.

1 Move the value from the memory word into a machine-register.

2. Increment the value in the machine register.

3. Move the new value into the memory word.

- Since the instruction sequence MOVE-ADD-MOVE may be used a number of times in a program, it is convenient to define a macro named INCR.
- Using lexical expansion the macro call INCR A,B,AREG can lead to the generation of a MOVE-ADD-MOVE instruction sequence to increment A by the value of B using AREG to perform the arithmetic.
- Use of Semantic expansion can enable the instruction sequence to be adapted to the types of A and B.
- For example an INC instruction could be generated if A is a byte operand and B has the value 1.
- If the parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement, then these parameters in macro definitions are called as positional parameters.

- Syntax and written as

  & < parameter name >

Example : &ROLLNO

where ROLLNO is the name of a parameter. In a positional parameter < parameter kind > is omitted. The < actual parameter spec > is a call on a macro using positional parameters is simply an < ordinary string >.

- The value of a positional formal parameter ABC is determined by the rule of positional association as follows :

1. In macro prototype statement, find original position of ABC.

2. Also find the actual parameter specification occupying the same ordinal position in the list of the actual parameters in the macro call statement.

Q.16 Explain concept of keyword parameters.

Ans. : Keyword Parameters :

For keyword parameters,

< parameter name > ⇒ ordinary string

< parameter kind > ⇒ string '='

< actual parameter spec > ⇒ written as

< formal parameter name > = < ordinary string >

- Value of formal parameter is determined as follows :

1. Find the actual parameter specification which look likes XYZ = < ordinary string >

2. < ordinary string > is in the specification of the string ABC. So formal parameter value is XYZ = ABC

Default specifications of parameters

- Standard assumption value is a default value. This is suitable in such condition that, the parameter has the same value in most calls. When the desired value is different from the default value, the desired value can be specified explicitly in a macro call. Syntax for formal parameter specification is as follows :

& < parameter name > [< parameter kind > [< default value >]]

Macros with mixed parameter lists

- Keyword parameters and positional parameters, both uses, a macro. In this situation, all positional parameters must precede all keyword parameters. For example :

  SUMUP X, Y, A = 10, B = n

- Here X and Y are positional parameters and A and B are keyword parameters and A and B are keyword parameters

6.8 : Introduction to Compilers

Q.17 What is compiler ?

Ans. : Definition : Compiler is a program which takes one language (source program) as input and translates it into an equivalent another language (target program).

- During this process of translation if some errors are encountered then compiler displays them as error messages.
- The basic model of compiler can be represented as in Fig. Q.17.1.

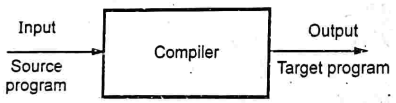


Fig. Q.17.1 Compiler

6.9 : Phases of Compiler

Q.18 Explain various phases of compiler.

Ans. : 1. Lexical Analysis :

- The lexical analysis is also called **scanning.**
- It is the phase of compilation in which the complete source code is scanned and your source program is broken up into group of strings called **token.**
- A token is a sequence of characters having a collective meaning.
- For example if some assignment statement in your source code is as follows,

total = count + rate * 10

- Then in lexical analysis phase this statement is broken up into series of tokens as follows.
  1. The identifier **total**
  2. The **assignment** symbol
  3. The identifier **count**
  4. The **plus** sign
  5. The identifier **rate**
  6. The **multiplication** sign
  7. The constant number **10**
- The **blank characters** which are used in the programming statement are eliminated during the lexical analysis phase.

2. Syntax Analysis :

- The syntax analysis is also called **parsing.**
- In this phase the tokens generated by the lexical analyser are grouped together to form a hierarchical structure.
- The syntax analysis determines the **structure of the source string** by grouping the tokens together.
- The hierarchical structure generated in this phase is called **parse tree** or **syntax tree.**
- For the expression **total = count + rate *10** the parse tree can be generated as shown in Fig. Q.18.1.

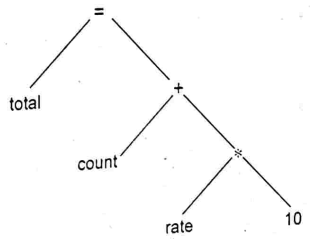


Fig. Q.18.1 Parse tree for total = count + rate*10

- In the statement '*total = count + rate *10* ' first of all *rate*10* will be considered because in arithmetic expression the multiplication operation should be performed before the

addition. And then the addition operation will be considered. For building such type of syntax tree the production rules are to be designed. The rules are usually expressed by **context free grammar.**

- For the above statement the production rules are -

(1) E → identifier
(2) E → number
(3) E → E1 + E2
(4) E → E1*E2
(5) E → (E)

where E stands for an expression.

- By rule (1) count and rate are expressions and
- by rule (2) 10 is also an expression.
- By rule (4) we get rate*10 as expression.
- And finally count + rate*10 is an expression.

3. Semantic Analysis

- Once the syntax is checked in the syntax analyser phase the next phase i.e. the semantic analysis determines the meaning of the source string.
- For example meaning of source string means matching of parenthesis in the expression, or matching of if ...else statements or performing arithmetic operations of the expressions that are type compatible, or checking the scope of operation.

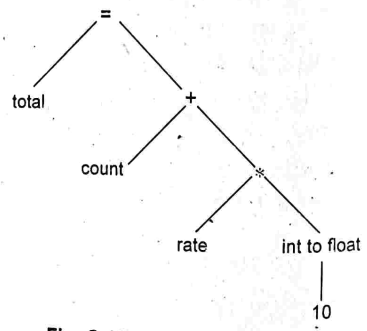


Fig. Q.18.2 Semantic analysis

4. Intermediate Code Generation

- The intermediate code is a kind of code which is easy to generate and this code can be easily converted to target code.
- This code is in variety of forms such as *three address code, quadruple, triple, posix.* Here we will consider an intermediate code in three address code form. This is like an assembly language.
- The three address code consists of instructions each of which has at the most three operands. For example,

t_1 := int to float (10)

t_2 := rate × t_1

t_3 := count + t_2

total := t_3

- There are certain **properties** which should be possessed by the three address code and those are,

1. Each three address instruction has at the most one operator in addition to the assignment. Thus the compiler has to decide the order of the operations devised by the three address code.
2. The compiler must generate a temporary name to hold the value computed by each instruction.
3. Some three address instructions may have fewer than three operands for example first and last instruction of above given three address code. i.e.

t_1 := int_to_float (10)

total := t_3

5. Code Optimization

- The code optimization phase attempts to improve the intermediate code.
- This is necessary to have a **faster executing code** or **less consumption of memory.**
- Thus by optimizing the code the overall running time of the target program can be improved.

6. Code Generation

- In code generation phase the target code gets generated. The intermediate code instructions are translated into sequence of machine instructions.

	Explanation of code
MOV rate, R1	Variable 'rate' is moved to register R1.
MUL #10.0, R1	Then R1 is multiplied with value 10 and result is stored in R1 ∴R1 = rate *10.
MOV count, R2	The value in 'count' is moved to reg. R2 then it is added with R1.
ADD R2, R1	∴R1 = count + rate *10.0
MOV R1, total	Finally contents of register R1 is moved to identifier 'total'.

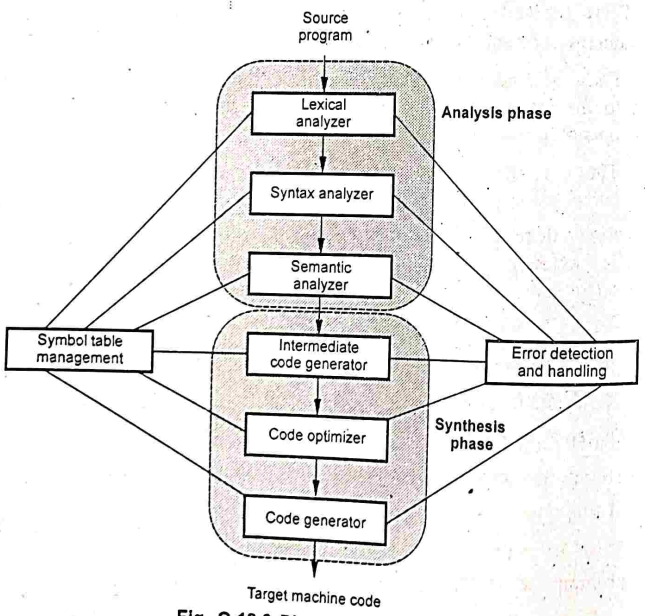


Fig. Q.18.3 Phases of compiler

Q.19 What is compiler? Explain any two phases of compiler with suitable diagram

☞ [SPPU : May-18, Marks 5]

Ans. : Compiler : Refer Q.17.

Phases of Compiler : Refer Q.18.

Q.20 Consider input "d = a + b * 2;" and show the output of each phase of compiler with suitable diagram.

☞ [SPPU : April-18, In Sem, Marks 6]

Ans. :

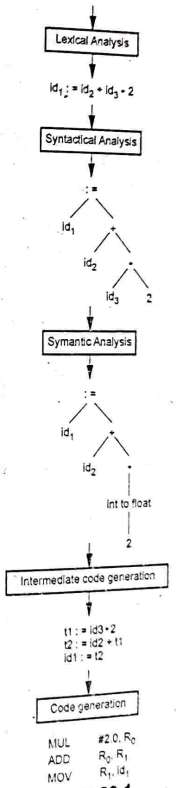


Fig. Q.20.1

6.10 : Loaders

Q.21 Define Loader.

Ans. : Loader is a utility program which takes object code as input prepares it for execution and loads the executable code into the memory. Thus loader is actually responsible for initiating the execution process.

Q.22 Discuss four different functions of loader.

☞ [SPPU : April-18, Marks 4]

OR What are types of loaders? Discuss four different functions of loaders. ☞ [SPPU : May-19, Marks 4]

Ans. : Types of Loaders : Based on various functionalities of the loader, there are various types of loaders, which are known as loading schemes Various schemes are -

(1) General Loader Scheme

(2) Subroutine Linkage

(3) Relocating Loaders

(4) Direct Linkage Loader

Functions of Loader :

The loader is responsible for the **activities** such as allocation, linking, relocation and loading

1) It allocates the space for program in the memory, by calculating the size of the program. This activity is called **allocation.**
2) It resolves the symbolic references (code/data) between the object modules by assigning all the user subroutine and library subroutine addresses. This activity is called **linking.**
3) There are some address dependent locations in the program, such address constants must be adjusted according to allocated space, such activity done by loader is called **relocation.**
4) Finally it places all the machine instructions and data of corresponding programs and subroutines into the memory. Thus program now becomes ready for execution, this activity is called **loading.**

6.11 : General Loader Scheme

Q.23 Explain "General loading scheme(using suitable diagram) with advantages and disadvantages.

☞ [SPPU : Dec.-18, March-20, Marks 5, April-19, Marks 6]

Ans. : • In this loader scheme, the source program is **converted to object program** by some translator (assmbler).

• The loader accepts these object modules and puts machine instruction and data in an executable form at their assigned memory.

• The loader occupies some portion of main memory.

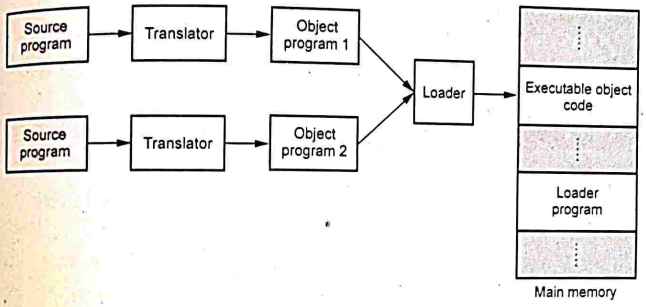


Fig. Q.23.1 General loader scheme

Advantages :

1. The program need not be retranslated each time while running it.
2. It is possible to write source program with multiple programs and multiple languages.

Disadvantages :

1. Some portion of memory is occupied by the loader program, hence memory is wasted.
2. If the program is modified it has to be retranslated.

6.12 : Subroutine Linkages

Q.24 What are subroutine linkages ? What are benefits of using it ?

☞ [SPPU : May-19, Marks 4]

Ans. : • A program consists of main program and set of functions(subroutines). These functions may reside in different files.

- The problem of subroutine linkage is : When a main program wants to call subroutine the subroutine B and if subroutine B resides in some another file then the assembler will not know the address of B and declare it as undefined symbol.
- To realize the interaction of A and B must contain public definition and external references.
- Subroutine linkage allows to establish the reference between main program and various subroutines residing in different files.

6.13 : Linkers

Q.25 Explain the concept of linkers.

Ans. : • During the process of linking the object module is created. This object module contains the target code and information about other programs and library routines that are required to call during the program execution.

- Linker extracts the information present in the object module, locates the required programs and library routines and combine them with the target code and produce a program which a machine can understand. Such a program is called binary program.

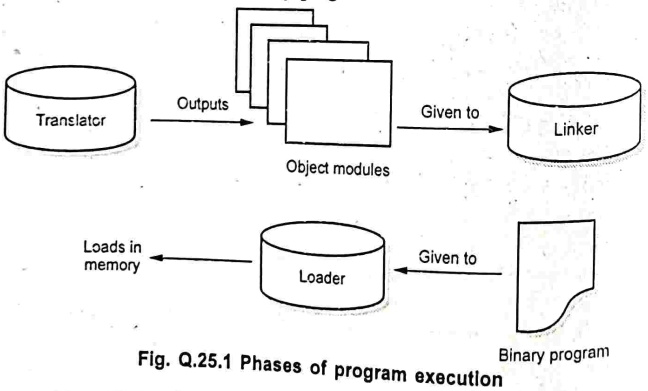


Fig. Q.25.1 Phases of program execution

- The loader then loads this binary program in the memory. (Refer Fig. Q.25.1)

6.14 : Relocation and Linking

Q.26 Explain relocation with suitable example.

Ans. : Program relocation is the process of changing the addresses used in the address sensitive instructions of a program so that the program can execute correctly from designated area of memory.

The relocation can be performed by either linker or loader -

1. If linked origin differs from the translated origin then relocation must be performed by the **linker.**
2. If load origin differs the linked origin, relocation must be performed by the **loader.**

The relocation factor can be calculated using following formula -

Relocation factor = Load origin – Translated origin

The translator puts the address of particular symbol which is used in the program. This is called the **translation time address ,**

Let A is the symbol used in some program then the translated time address is denoted as -

Translated time address = Translated origin + Offset of A

Similarly the **linked time address** can be denoted as

Linked time address = Linked origin + Offset of A

Another way of denoting linked time address is

Linked time address = Translated time address + Relocation factor

Thus relocation of the program X is the sum of relocation factor for X and translation time address for every instruction.

Example of Relocation

Suppose there is a program A which contains the set of instructions which require relocation. Let us name this set as X.

The relocation of program A can be performed by computing the relocation factor and adding it to **translation time** address.

Following information is available -

1. Suppose there is a command **linker 800 A, B**
   Then in this **linker** command 800 denotes the **load origin.**
2. Let for the set of instructions X (of program A) that needs to be relocated the **relocation table** is as follows.

Translated Address
400
438

3. The instruction with **translated address** 400 contains the address 330 in operand field.
4. The instruction with translated address 438 contain the address 302 in operand field translated origin factor.
5. **Calculate relocation factor**
   Relocation factor = Load origin – Translated origin
   = 800 – 500
   Relocation factor = 300

1) For instruction with translated address 400 which contains address 330 in operand field, the relocation address will be -
   Relocation factor + Address in operand field
   = 300 + 330 = **630**
2) For instruction with translated address 438 which contains address 302 in operand field, the relocation address will be -
   Relocation factor + Address in operand field
   = 300 + 302 = **602**

Thus relocatable addresses are now **630** and **602.**

Q.27 What is object module ?

Ans. : The object module is nothing but sequence of records. The purpose of object record is to describe specific aspect of the program.

END...

SOLVED MODEL QUESTION PAPER (In Sem)
Operating Systems
T.E. (IT) Semester - V [As Per 2019 Pattern]

Time : 1 Hour] **[Maximum Marks : 30**

N.B. : i) Attempt Q.1 or Q.2, Q.3 or Q.4.
ii) Neat diagrams must be drawn wherever necessary.
iii) Figures to the right side indicate full marks.
iv) Assume suitable data, if necessary.

Q.1 a) *What is kernel ? List different types of kernel.*
(Refer Q.8 of Chapter - 1) **[4]**

b) *What are the benefits of resource abstraction ?*
(Refer Q.3 of Chapter - 1) **[3]**

c) *List and explain components of Linux.*
(Refer Q.13 of Chapter - 1) **[8]**

OR

Q.2 a) *Describe in detail the functions of OS as a resource manager.*
(Refer Q.2 of Chapter - 1) **[5]**

b) *State and explain different services provided by an operating system.* **(Refer Q.11 of Chapter - 1)** **[4]**

c) *Explain the significance of following shell commands*
i) in ii) wc iii) umask iv) cut v) grep
(Refer Q.15 of Chapter - 1) **[6]**

Q.3 a) *Define process. Differentiate between a process and a program.*
(Refer Q.1 of Chapter - 2) **[3]**

b) *What should be scheduling criteria for scheduling algorithm ?* **(Refer Q.23 of Chapter - 2)** [4]

c) *For the table given below calculate average waiting time and average turnaround time and draw a Gantt Chart illustrating the process execution using following scheduling algorithms.*
i) RR (Time slice - 2 units) ii) SJF (non - preemptive)

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	5
P_3	3	3
P_4	4	1
P_5	6	4

(Refer Q.26 of Chapter - 2) [8]

OR

Q.4 a) *Explain thread life cycle.* **(Refer Q.9 of Chapter - 2)** [5]

b) *Explain the following functions with reference to 'C'.*
i) pthread_create () ii) pthread_join() **(Refer Q.17 of Chapter - 2)** [4]

c) *Explain kernel level thread with their advantages and disadvantages.* **(Refer Q.15 of Chapter - 2)** [6]

SOLVED MODEL QUESTION PAPER (End Sem)
Operating Systems
T.E. (IT) Semester - V [As Per 2019 Pattern]

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

N.B. : ***i) Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***
ii) Neat diagrams must be drawn wherever necessary.
iii) Figures to the right side indicate full marks.
iv) Assume suitable data, if necessary.

Q.1 a) *Explain the following terms :*
i) Mutual exclusion ii) Synchronization iii) Race condition
(Refer Q.4 of Chapter - 3) [6]

b) *State and explain the deadlock recovery methods.*
(Refer Q.34 of Chapter - 3) [6]

c) *Explain with an appropriate example, how resource allocation graph determines a deadlock.* **(Refer Q.26 of Chapter - 3)** [6]

OR

Q.2 a) *Write a pseudo code for producer-consumer problem using semaphore.* **(Refer Q.17 of Chapter - 3)** [6]

b) *Consider the following snapshot of the system.*

	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			

P3	3	0	2	9	0	2		
P4	2	1	1	2	2	2		
P5	0	0	2	4	3	3		

Answer the following questions using Banker's algorithm.

a) What are the values of Need Matrix ?

b) Is the system in the safe mode ? If yes, what is the safe sequence ?

c) If a request from process P1 arrives for (1, 0, 2), can the request be granted immediately ? **(Refer Q.31 of Chapter - 3)** [12]

Q.3 a) *What is fragmentation ? Explain types of fragmentation.* **(Refer Q.7 of Chapter - 4)** [5]

b) *For the following reference string.*

5,6,7,8,5,6, 9,5,6,7,8,9,6,7,4,9,8

Count the number of page fault occur with 3 frames using FIFO, optimal and LRU page replacement methods. Discuss the result. **(Refer Q.27 of Chapter - 4)** [12]

OR

Q.4 a) *What is paging ? Explain in details.* **(Refer Q.14 of Chapter - 4)** [5]

b) *For the following reference string. 6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 6, 3, 2, 1, 2 Count the number of page faults that occur with 3 frames using FIFO, Optimal and LRU page replacement methods. Discuss the result.* **(Refer Q.28 of Chapter - 4)** [12]

Q.5 a) *Suppose that a disk drive has 400 cylinders, numbered 0 to 399. The drive is currently serving a request at cylinder 160. The queue of pending requests, in FIFO order, is 370, 30, 390, 130, 310, 170, 340, 180. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms ?*

i) FCFS ii) SSTF iii) SCAN iv) LOOK

(Assume that the head is initially moving towards track number 399). **(Refer Q.13 of Chapter - 5)** [12]

b) *Discuss the following with respect to file system : file operation* **(Refer Q.15 of Chapter - 5)** [6]

OR

Q.6 a) *Explain SSTF and LOOK disk scheduling algorithm.* **(Refer Q.7 of Chapter - 5)** [8]

b) *Draw and explain file system architecture.* **(Refer Q.17 of Chapter - 5)** [6]

c) *Enlist the characteristics of block and character devices? Explain each with suitable example.* **(Refer Q.1 of Chapter - 5)** [4]

Q.7 a) *With syntax and example types of assembly language statements.* **(Refer Q.5 of Chapter - 6)** [6]

b) *Explain "General loading scheme(using suitable diagram) with advantages and disadvantages.* **(Refer Q.23 of Chapter - 6)** [6]

c) *Explain relocation with suitable example.* **(Refer Q.26 of Chapter - 6)** [5]

OR

Q.8 a) *Explain in detail Operation code table and symbol table.* **(Refer Q.9 of Chapter - 6)** [6]

b) *Explain various phases of compiler.* **(Refer Q.18 of Chapter - 6)** [8]

c) *Explain task of analysis phase and synthesis phase.* **(Refer Q.7 of Chapter - 6)** [3]

END...

DECODE®

A Guide For Engineering Students

OPERATING SYSTEMS

(For END SEM Exam - 70 Marks)

SUBJECT CODE : 314442

T.E. (Information Technology) Semester - V

© Copyright with Technical Publications
All publishing rights (printed and ebook version) reserved with Technical Publications.
No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :

TECHNICAL PUBLICATIONS®
SINCE 1992 An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA Ph.: +91-020-24495496/97
Email : info@technicalpublications.in
Website : www.technicalpublications.in

Printer :
Yogiraj Printers & Binders, Sr.No. 10/1A,Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-5585-078-2

9 789355 850782

SPPU 19

9789355850782 [2]

JUNE - 2022 [END SEM] [5870] - 1141 **Solved Paper**

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instructions to the candidates :

1) Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.

2) Neat diagram must be drawn wherever necessary.

3) Figures to the right indicate full marks.

4) Assume suitable data, if necessary.

Q.1 a) *Explain requirements for mutual exclusion.*
(Refer Q.3 of Chapter - 3) **[8]**

b) *Explain following terms :*
i) Principles of concurrency. **(Refer Q.1 of Chapter - 3)**
ii) Semaphores. **(Refer Q.9 of Chapter - 3)**
iii) Mutex **[9]**

Ans. : iii) Mutex :

- Mutex is used to ensure only one thread at a time can access the resource protected by the mutex. The process that locks the mutex must be the one that unlocks it. Mutex is good only for managing mutual exclusion to some shared resource.
- Mutex is easy and efficient for implement.
- Mutex can be in one of two states : Locked or unlocked.
- Mutex is represented by one bit. Zero (0) means unlock and any other value represents locked. It uses two procedures.

OR

Q.2 a) *Explain following synchronization problems*
i) Readers / Writers problem. **(Refer Q.15 of Chapter - 3)**
ii) Producer and consumer problem. **(Refer Q.19 of Chapter - 3)** **[8]**

b) *Explain inter-process communication using*

i) Pipes **(Refer Q.20 of Chapter - 3)**

ii) Shared memory **(Refer Q.20 of Chapter - 3)**

iii) Semaphore. **(Refer Q.9 of Chapter - 3)** **[9]**

Q.3 a) *For the given reference string with 4 page frame abvailable, determine the number of page faults for FIFO, OPTIMAL, LRU algorithms :*

3, 5, 3, 7, 2, 1, 5, 4, 6, 7, 4, 1, 2. **[12]**

Ans. : FIFO

	3	5	3	7	2	1	5	4	6	7	4	1	2
1	3	3	3	3	3	1	1	1	1	1	1	1	2
2	-	5	5	5	5	5	5	4	4	4	4	4	4
3	-	-	-	7	7	7	7	7	6	6	6	6	6
4	-	-	-	-	2	2	2	2	2	7	7	7	7
Page fault	PF	PF		PF	PF	PF		PF	PF	PF			PF

Total number of page fault in FIFO = 9

Optimal

	3	5	3	7	2	1	5	4	6	7	4	1	2
1	3	3	3	3	3	1	1	1	1	1	1	1	1
2	-	5	5	5	5	5	5	4	4	4	4	4	2
3	-	-	-	7	7	7	7	7	6	6	6	6	6
4	-	-	-	-	2	2	2	2	2	7	7	7	7
Page fault	PF	PF		PF	PF	PF		PF	PF	PF			PF

Total number of page fault in optimal = 9

LRU

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

3	-	-	-	7	7	7	7	4	4	4	4	4	4
4	-	-	-	-	2	2	2	2	6	6	6	6	2
Page fault	PF	PF		PF	PF	PF	PF	PF	PF	PF		PF	PF

Total number of page fault in LRU = **11**

b) *Explain with the help of neat diagram hierarchical page table.* **(Refer Q.15 of Chapter - 4)** **[6]**

OR

Q.4 a) *A1 MByte block of memory is allocated using the buddy system.*

i) Show the result of the following sequence in the graphical form for A : Request 35 KB, B : Request 140 KB, C : Request 120 KB, D : Request 250 KB, return C, E : Request 70 KB, return B, return D, F : Request, 100 KB, return A, return E.

ii) Draw the tree representation after return B. **[10]**

Ans. : i)

	0 128 K	256 K	384 K	512 K	768 K	1 M
Initially						
Request 35 KB	A	128	256			
Request 140 KB	A	128	B			
Request 128 KB	A	C	B			
Request 250 KB	A	C	B	D	256 KB	
Return C	A	128	B	D	256 KB	
Request 70	A E	128	B	D	256 KB	
Return B, D	A E	128	256	512		
Request 100 KB	A E	F	256	512		
	128	F	256	512		

ii)

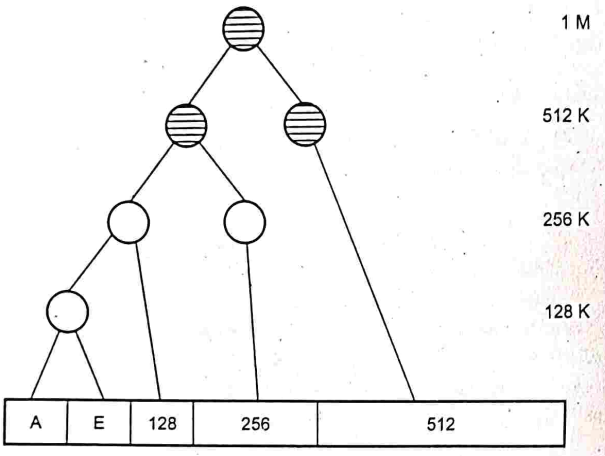


Fig. 1

b) *What is principle of locality ? Explain working set model.* **[8]**

Ans. : Working sets model is proposed by Peter Denning to prevent thrashing.

- We define the working set of information W(t, τ) of a process at time t to be the collection of information referenced by the process during the process time interval (t – τ, t).

  Fig. 2 shows working set model.

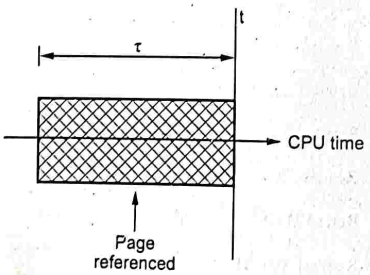


Fig. 2 Working set model

- Here τ is the working set parameter. The elements of W(t, τ) are pages but they can be anything else used by a process.

- If the sum of all working sets of all runnable threads exceeds the size of memory, then stop running some of the threads for a while. Divide processes into two groups : Active and inactive.
- When a process is active its entire working set must always be in memory : Never execute a thread whose working set is not resident. When a process becomes inactive, its working set can migrate to disk.
- Threads from inactive processes are never scheduled for execution. The collection of active processes is called the balance set. The system must have a mechanism for gradually moving processes into and out of the balance set. As working sets change, the balance set must be adjusted.

Also refer Q.31 of Chapter - 4.

Q.5 a) *Assume a disk with 200 tracks and the disk request queue has random requests in it as follows : 55, 58, 39, 18, 90, 160, 150, 38, 184. Find the no of tracks traversed and average seek length if*

i) FCFS ii) SSTF iii) SCAN

iv) C-Look is used and initially head is at track no 100. Assume head is moving towards outer track for SCAN and C-Look. **[12]**

Ans. :

Disk request	FCFS		SSTF		SCAN		C-SCAN	
	Next track	No. of track traver sed	Next track	No. of track traver sed	Next track	No. of track traver sed	Next track	No. of track traver sed
55	55	45	90	10	150	50	150	50
58	58	03	58	32	160	10	160	10
39	39	19	55	03	184	24	184	24
18	18	21	39	16	90	94	18	166
90	90	72	38	01	58	32	38	20
160	160	70	18	31	55	03	39	01
150	150	10	150	132	39	16	55	16
38	38	112	160	10	38	01	58	03
184	184	146	184	24	18	20	90	32
Total track traversed		498		259		250		322

b) *State and explain diff approaches of I/O buffering.* [6]
(Refer Q.5 of Chapter - 5)

OR

Q.6 a) *Describe three methods of record blocking with the help of neat diagrams.* **(Refer Q.18 of Chapter - 5)** [9]

b) *Explain different file organization techniques.* [9]
(Refer Q.23 of Chapter - 5)

Q.7 a) *Explain working of "general loading scheme" with advantages and disadvantages.* **(Refer Q.23 of Chapter - 6)** [6]

b) *Draw and explain flowchart of pass-2 of two pass assembler.* [6]

Ans. :

- Fig. 3 shows flowchart of Pass-2 of two pass assembler.

See Fig. 3 on next page.

- Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration (machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration.
- During the second pass, the assembler examines the operands for symbolic references to storage locations and resolves these symbolic references using information in the symbol table.
- On the second pass, the assembler :

1) Examines the operands for symbolic references to storage locations and resolves these symbolic references using information in the symbol table.
2) Ensures that no instructions contain an invalid instruction form.
3) Translates source statements into machine code and constants, thus filling the allocated space with object code.
4) Produces a file containing error messages, if any have occurred.

Flowchart for Pass 2

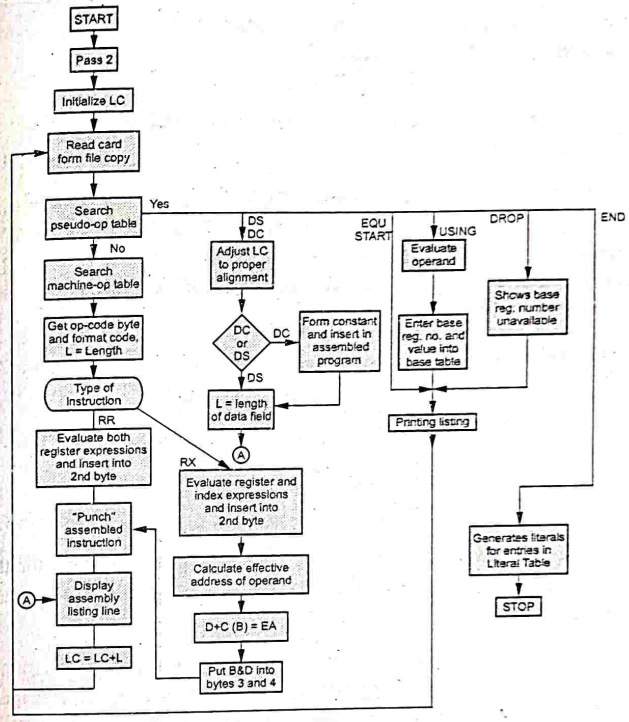


Fig. 3 Pass 2 flowchart

c) *Define following system software components with suitable diagram.*

i) Compiler **(Refer Q.17 of Chapter - 6)**
ii) Loader **(Refer Q.21 of Chapter - 6)**
iii) Editor
iv) Linker **(Refer Q.25 of Chapter - 6)**
v) Debugger **(Refer Q.2 of Chapter - 6)** [5]

Ans. : iii) Editor : In general, an editor refers to any program capable of editing files. A text editor refers to any form of computer program that enables users to create, change, edit, open and view plain text files.

Also refer Q.2 and Q.3 of Chapter - 6.

OR

Q.8 a) *Explain phase structure of compiler with neat diagram.* **[6]**

(Refer Q.18 of Chapter - 6)

b) *What is the need of Symbol Table (ST) and Literal Table (LT) in two pass assembler ? Explain fields of ST and LT with suitable example.* **[6]**

Ans. : • Literal table is used for keeping track of literals that are encountered in the programs.

Literal Table (LITTAB)

	Literal	Address
1	= '5'	
2	= '1'	
3	= '1'	

- We directly specify the value, literal is used to give a location for the value. Literals are always encountered in the operand field of an instruction.
- Literals are relocatable terms because the address of the literal, rather than the literal-generated constant itself, is assembled in the statement that references a literal. The assembler generates the literals, collects them and places them in a specific area of storage, as explained under literal pool.
- Also refer Q.9 of Chapter - 6.

c) *What are types of loaders ? Discuss four different functions of loaders.* **(Refer Q.22 of Chapter - 6)** **[5]**

DECEMBER - 2022 [END SEM] [5926] - 120 **Solved Paper**

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Q.1 a) *What is semaphore and mutex ? Explain with the help of pseudocode, how semaphore is used to solve producer consumer problem ?* **(Refer Q.9 and Q.17 of Chapter - 3, Mutex : Q.1 (b) of June-2022)** **[9]**

b) *What are the four necessary conditions for deadlock ? How is a deadlock detected in a system with resources having single instances ? Explain with an example.* **(Refer Q.24 and Q.26 of Chapter - 3)** **[9]**

OR

Q.2 a) *Define mutual exclusion, race condition, semaphore. Deadlock.* **[6]**

Ans. : Mutual exclusion, race condition : Refer Q.4 of Chapter - 3.

Semaphore : Refer Q.9 of Chapter - 3.

Deadlock : Refer Q.23 of Chapter - 3.

b) *What is bankers safe sequence algorithm ? Apply it for finding safe sequence of execution of 5 processes in a system having snapshot at time T_0 :* **[12]**

(Refer similar Q.31 of Chapter - 3)

	Allocation	*Max*	*Available*
	A B C	*A B*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Also determine whether following requests can be granted or not :

i) Request for process P_2 : 3 0 0 and
ii) Request for process P_3 : 0 0 1

Q.3 a) *Given memory partitions of 150 K, 650 K, 280 K, 390 K and 540 K (in order) how would each of the first fit, best fit and worst fit algorithms place processes of 212 K, 457 K, 112 K, 510 K and 326 K (in order).* **[9]**

Ans. : First-fit :

212 K is put in 650 K partition.

457 K is put in 540 K partition.

112 K is put in 438 K partition (new partition 438 K = 650 K - 212 K).

510 K must wait.

326 K is put in 390 K partition.

Best-fit :

212 K is put in 280 K partition.

457 K is put in 540 K partition.

112 K is put in 150 K partition.

510 K is put in 650 K partition.

326 K is put in 390 K partition.

Worst-fit :

212 K is put in 650 K partition.

457 K is put in 540 K partition.

112 K is put in 390 K partition.

510 K must wait.

326 K is put in 438 K partition (new partition 438 K = 650 K - 212 K).

b) *With the help of neat diagrams, write a short note on buddy system.* **(Refer Q.13 of Chapter - 4)** **[8]**

OR

Q.4 a) *Explain Belady's anomaly with suitable example.* **[6]**

Ans. : • Using FIFO algorithm for some reference strings actually generate more page faults when more page frames are allotted. This truly unexpected result was first demonstrated by Belady in 1970 and is known as Belady's anomaly.

• FIFO does not satisfy the inclusion property and is not a stack algorithm.

• A page replacement algorithm that satisfies the inclusion property is free from Belady anomaly.

• Consider the reference string : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

• Page frame size = 3

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Page frame 2	-	2	2	2	1	1	1	1	1	3	3	3
Page frame 3	-	-	3	3	3	2	2	2	2	2	4	4
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	

Total number of page fault = 9

Page frame size = 4

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Page frame 2	-	2	2	2	2	2	2	1	1	1	1	5
Page frame 3	-	-	3	3	3	3	3	3	2	2	2	2
Page frame 4	-	-	-	4	4	4	4	4	4	3	3	3
Page fault	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

Total number of page fault = 10

- Paging algorithm has worse performance when the amount of primary memory allocated to the process is increased. The problem creates because the set of pages loaded with a memory allocation of 3 is not same as the with a memory allocation of 4.
- There are a set of demand paging algorithms whereby the set of pages loaded with an allocation of m frames is always a subset of the set of pages loaded with an allocation of m + 1 frame. This property is called the inclusion property.

b) *Consider the following segment table :* **[6]**

Segment	*Base*	*Length*
0	1790	350
1	2722	1050
2	520	925
3	5200	450
4	4200	655

What are the physical addresses for the following logical address ?

i) 0, 330 ii) 2,525 iii) 4,700
iv) 3,400 v) 1,1110.

Ans. : I) 0, 330

Offset = 330; Segment length = 350; Segment = 0

Offset 330 is less than segment length 350. Starting location of segment 0 is start from 1790.

Physical address = Offset + Segment base = 330 + 1790 = 2120

II) 2,525

Offset = 525; Segment length = 1050; Segment = 1

Offset 525 is less than segment length 1050. Starting location of segment 1 is start from 2722.

Physical address = Offset + Segment base = 525 + 2722 = 3247

III) 4,700

Offset = 700; Segment length = 925; Segment = 2

Offset 700 is less than segment length 925. Starting location of segment 2 is start from 520.

Physical address = Offset + Segment base = 700 + 520 = 1220

Iv) 3, 400

Offset = 400; Segment length = 450; Segment = 3

Offset 400 is less than segment length 450. Starting location of segment 3 is start from 5200.

Physical address = Offset + Segment base = 400 + 5200 = 5600

v) 1, 1110

Offset = 1110; Segment length = 655; Segment = 4

Offset 1110 is greater than segment length 655. This address result in segment fault.

c) *What are the distinctions among logical, relative and physical addresses ?* **[5]**

Ans. : • **Logical address** is generated by the CPU. This address is also called virtual address.

- Main memory address uses **physical address**. This address also called real address.
- **Logical address space :** Set of all logical addresses generated by a program.
- Logical address and physical address is identical when load time and compile time address binding is performed. The execution time address binding generates different physical and logical address.
- Memory Management Unit (MMU) is responsible for run time address mapping from virtual to physical address.
- Relative address means an address specified by indicating its distance from another address, called the **base address**. Relative addressing is always in bytes.

Q.5 a) *A disk drive has 200 tracks, numbered 0-199. The drive is currently serving the request at track no 53. The queue of pending requests in FIFO order is 98, 183, 37, 122, 14, 124, 65, 67. Starting from the current head position what is the total distance that disk arm moves to satisfy all the pending requests for the following disk scheduling algorithms. Assume that the head is moving in the increasing order of track number for SCAN and C-LOOK.*

i) FCFS ii) SCAN

iii) C-LOOK iv) SSTF. **(Refer Q.9 of Chapter - 5)** **[12]**

b) *Explain with diagrams different I/O buffering techniques.* **(Refer Q.5 of Chapter - 5)** **[6]**

OR

Q.6 a) *List and explain different file access methods.* **(Refer Q.18 of Chapter - 5)** **[9]**

b) *Describe different methods of record blocking with the help of a neat diagram.* **(Refer Q.23 of Chapter - 5)** **[9]**

Q.7 a) *What is system software ? Explain any four system software in brief ?* **(Refer Q.1, Q.2 and Q.3 of Chapter - 6)** **[6]**

b) *Explain imperative statement, declarative statement, and assembly directive of assembly language programming ?* **(Refer Q.5 of Chapter - 6)** **[6]**

c) *Discuss with example what is forward reference problem.* **(Refer Q.11 of Chapter - 6)** **[5]**

OR

Q.8 a) *Explain ORIGIN, EQU and LTROG with an example.* **(Refer Q.10 of Chapter - 6)** **[6]**

b) *Explain the data structures required for two PASS assembler in detail.* **[6]**

Ans. : Fig. 1 shows the data structure used by the analysis and synthesis phases.

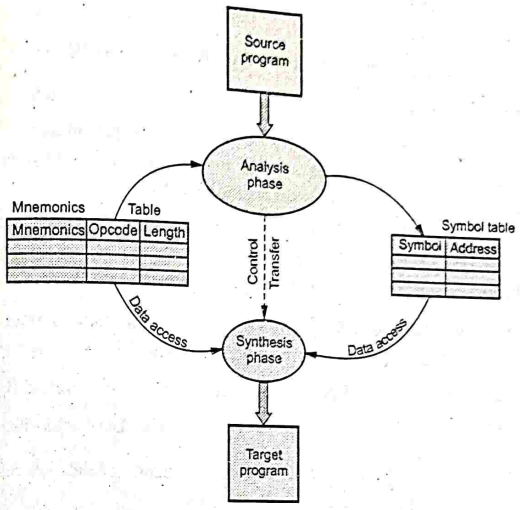


Fig. 1 Data structure with analysis and synthesis phase

- Mnemonic table is a fixed table and only used in the analysis and synthesis phases.
- Symbol table is constructed during analysis phase and used during synthesis.

Analysis phase task :

1. Separate the label, mnemonic opcode and operand fields of a statement.
2. If statement contains label then the pair of (symbol, < LC content >) is entered in the symbol table.
3. Validity of the mnemonic opcode is checked through a look-up table by using mnemonics table.
4. Location counter processing is done.

Synthesis phase task :

1. Read the machine opcode corresponding to the mnemonic from the mnemonics table.
2. Get an address of a memory operand from the symbol table.
3. Synthesize a machine instruction.

c) *Differentiate between literal and immediate operand.* **[5]**

Ans. : Let the programmers to be able to write the value of a constant operand as a part of the instruction that uses it. This avoids having to define the constant elsewhere in the program and make up a label for it. Such an operand is called a **literal.**

- For literals prefix is =, and for immediate addressing prefix is #.
- In immediate addressing, the operand value is assembled as part of the machine instruction, i.e. there is no memory reference.

Line no	Location counter	
55	0020	LDA #03 010003

- In the above example the last 12 bits of the machine code corresponds to 003 which is equal to the immediate value.
- A literal is identified with the prefix =, which followed by a specification of the literal value. For example

45 001A ENDFIL LDA = C' EOF' 032010

Specifies a 3-byte operand with value 'EOF'.

JUNE - 2023 [END SEM] [6003] - 702 Solved Paper

Course 2019

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Q.1 a) *What conditions are generally associated with readers-writers problem ? Write its pseudo code.* **(Refer Q.15 of Chapter - 3)** **[9]**

b) *Describe resource allocation graph in detail.* **(Refer Q.26 of Chapter - 3)** **[9]**

OR

Q.2 a) *Enlist different IPC techniques. Differentiate between named pipe and unnamed pipe with suitable example.* **(Refer Q.20 and Q.21 of Chapter - 3)** **[9]**

b) *What is critical section problem ? Give semaphore solution for producer consumer problem.* **(Refer Q.19 of Chapter - 3)** **[9]**

Q.3 a) *Consider six memory partitions of size 100 KB, 300 KB, 50 KB, 200 KB, 150 KB and 200 KB. These partitions need to be allocated to processes of sizes 200 KB, 100 KB, 50 KB in that order. Perform the allocation of processes using dynamic partitioning algorithms given below and comment on internal and external fragmentation -*

i) First fit algorithm ii) Best fit algorithm iii) Worst fit algorithm. **[12]**

Ans. : i) First fit algorithm

200 KB is put in 300 KB

100 KB is put in 100 KB (300 KB - 200 KB)

50 KB is put in 50 KB

ii) Best fit algorithm

200 KB is put in 200 KB

100 KB is put in 100 KB

50 KB is put in 50 KB

iii) Worst fit algorithm

200 KB is put in 300 KB

100 KB is put in 200 KB

50 KB is put in 200 KB

b) *Explain buddy system memory allocation with suitable example.* **(Refer Q.3 (b) of Dec.-2022)** **[5]**

OR

Q.4 a) *Find the number of page faults for the reference string 6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 6, 3, 2, 1, 2 using FIFO, LRU and optimal page replacement strategies. Consider frame size as 3.*

(Refer Q.28 of Chapter - 4) **[12]**

b) *Explain Belady's anomaly with suitable example.*

(Refer Q.4 (a) of Dec.-2022) **[5]**

Q.5 a) *Assume a disk with 200 tracks and the disk request queue has random requests in it as follows : 55, 58, 39, 18, 90, 160, 150, 38, 184. Find the no of tracks traversed and average seek length if*
1) SSTF 2) SCAN 3) C-SCAN.
Is used and initially head is at track no 100. **[12]**

Ans. :

Disk request	FCFS		SSTF		SCAN		C-SCAN	
	Next track	No. of track traversed	Next track	No. of track traversed	Next track	No. of track traversed	Next track	No. of track traversed
55	55	45	90	10	150	50	150	50
58	58	03	58	32	160	10	160	10
39	39	19	55	03	184	24	184	24
18	18	21	39	16	90	94	18	166
90	90	72	38	01	58	32	38	20
160	160	70	18	31	55	03	39	01
150	150	10	150	132	39	16	55	16
38	38	112	160	10	38	01	58	03
184	184	146	184	24	18	20	90	32
Total track traversed		498		259		250		322

1) SSTF

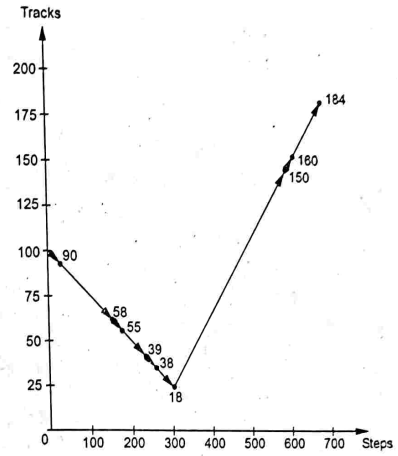


Fig. 1

2) SCAN

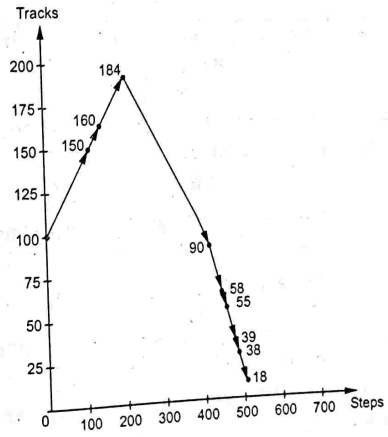


Fig. 2

3) C-SCAN

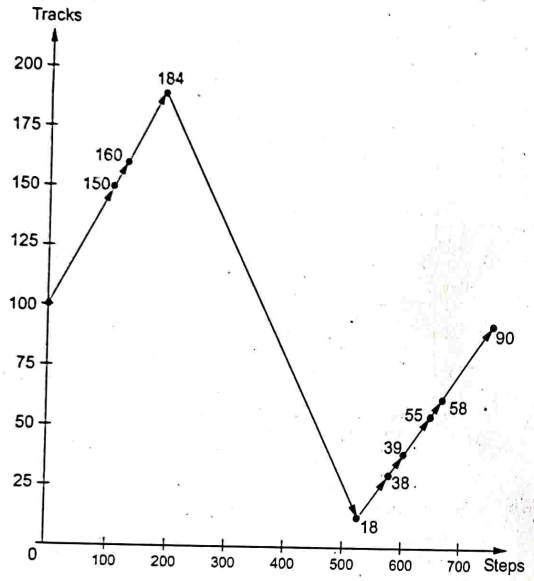


Fig. 3

b) *What are typical operations that may be performed on a directory ?* **(Refer Q.19 of Chapter - 5)** **[6]**

OR

Q.6 a) *What is I/O buffering ? Why I/O buffering is needed ? State and explain different approaches of I/O buffering.* **(Refer Q.4 and Q.5 of Chapter - 5)** **[9]**

b) *Explain with example any three disk scheduling criteria.* **[9]**

Ans. : Refer Q.6 of Chapter - 5.

- **Access time :** The sum of the seek time and the rotational delay equals the **access time.**

Access time = Seek time + Rotational latency

- **Transfer time :**

$$\text{Transfer time} = \frac{\text{Number of bytes to be transferred}}{\text{Number of bytes on the track} \times \text{Rotation speed}}$$

Q.7 a) *List down the phases of a compiler. Explain with suitable example.* **(Refer Q.18 of Chapter - 6)** **[9]**

b) *Explain macro call and macro expansion with suitable example.* **[8]**

Ans. : The occurrence of the macro names in the source program is known as macro call. Whenever there is a macro call, the macro processor expands those calls by substituting the group of statements of that specific macro.

Sample program	Meaning	Expanded code
MACRO	Start of macro definition	
FIRST	Macro name	
A 1, DATA B 2, DATA	Body of macro	
MEND	End of macro definition	
---------	Other source program lines	

FIRST	Macro call	A 1, DATA B 2, DATA
DATA DC F'10'	Defining DATA as constant Value of 10 fullword	DATA DC F'10'
---------		---------

- The directive MACRO, MEND, macro name and macro definition do not appear in expanded code.

- A macro call leads to macro expansion. During macro expansion, the macro call statement is replaced by a sequence of assembly statements.
- To differentiate between the original statements of a program and the statements resulting from macro expansion, each expanded statement is marked with " + " preceding its label field.
- Two key notions concerning macro expansion are :

1. **Expansion time control flow :** This determines the order in which model statements are visited during macro expansion.
2. **Lexical substitution :** Lexical substitution is used to generate an assembly statement from model statement.

Also refer Q.15 of Chapter - 6.

OR

Q.8 a) *Explain with example imperative statement, declarative statement and assembly directive of assembly language programming ?* **(Refer Q.5 of Chapter - 6)** **[9]**

b) *What is system software explain any four system software in brief ?* **(Refer Q.1, Q.2 and Q.3 of Chapter - 6)** **[8]**

END...