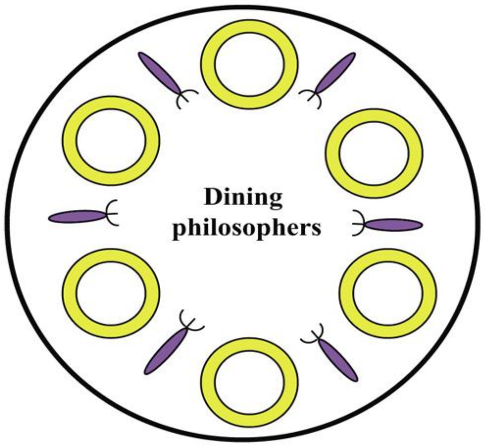
I^st Batch

Section A $(2 \times 10 = 20)$

1. Define the term semaphore. How does semaphore help in dining philosophers problem?

Semaphore are the integer variable. They could have value 0, indicating that no wake ups were saved or some positive value if one or more wakeups were pending. In these method two operations down and up are used instead of sleep and wake up operations. The down operation on a semaphore checks to see if the value is greater than 0. If so, it decrements and just continue. If the value is 0, the process is put to sleep without completing the down for the moment. Checking the value changing it and possibly going to sleep is all done as a single indivisible atomic action. The up operation increases the value of semaphore addressed. If one or more process were sleeping on that semaphore, unable to complete on earlier down operation, one of them is chosen by the system and is allowed to complete its down.

In 1965, Dijkstra posed and solved a synchronization problem called the dining philosophers problem. The 5 philosophers around a table and 5 forks. Philosophers eat/think. Eating needs 2 forks. Pick one fork at a time. The life of philosopher consists of alternative period of eating and thinking. When philosopher gets hungry, he/she tries to acquire her left and right forks; he/she eats for a while, then puts down the forks and continues to think.



When the philosopher is hungry, he/she picks up a fork and waits for another fork, when gets it eats for a while and put both forks back to the table.

A solution to the dining philosophers problem is given below:

```
#define N 5                  /* number of philosophers */
#define LEFT (i+N−1)%N       /* number of i's left neighbor */
#define RIGHT (i+1)%N        /* number of i's right neighbor */
#define THINKING 0           /* philosopher is thinking */
```

```
#define HUNGRY 1                /* philosopher is trying to get forks */
#define EATING 2                /* philosopher is eating */
typedef int semaphore;          /* semaphores are a special kind of int */
int state[N];                   /* array to keep track of everyone's state */
semaphore mutex = 1;            /* mutual exclusion for critical regions */
semaphore s[N];                 /* one semaphore per philosopher */
void philosopher (int i)        /* i: philosopher number, from 0 to N−1 */
{       while (TRUE)
        {                       /* repeat forever */
                think();        /* philosopher is thinking */
                take_forks(i);  /* acquire two forks or block */
                eat();          /* yum-yum, spaghetti */
                put_forks(i);   /* put both forks back on table */
        }
}
void take_forks(int i)          /* i: philosopher number, from 0 to N−1 */
{       down(&mutex);           /* enter critical region */
        state[i] = HUNGRY;      /* record fact that philosopher i is hungry */
        test(i);                /* try to acquire 2 forks */
        up(&mutex);             /* exit critical region */
        down(&s[i]);            /* block if forks were not acquired */
}
void put_forks(i)               /* i: philosopher number, from 0 to N−1 */
{       down(&mutex);           /* enter critical region */
        state[i] = THINKING;    /* philosopher has finished eating */
        test(LEFT);             /* see if left neighbor can now eat */
        test(RIGHT);            /* see if right neighbor can now eat */
        up(&mutex);             /* exit critical region */
}

void test(i)                         /* i: philosopher number, from 0 to N−1 */
{       if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
        {       state[i] = EATING;
                up(&s[i]);
        }
}
```

2. Explain how file allocation table (FAT) manages the files. Mention the merits and demerits of FAT system. A 200 GB disk has 1-KB block size, calculate the size of the file allocation table if each entry of the table to be 3 bytes.

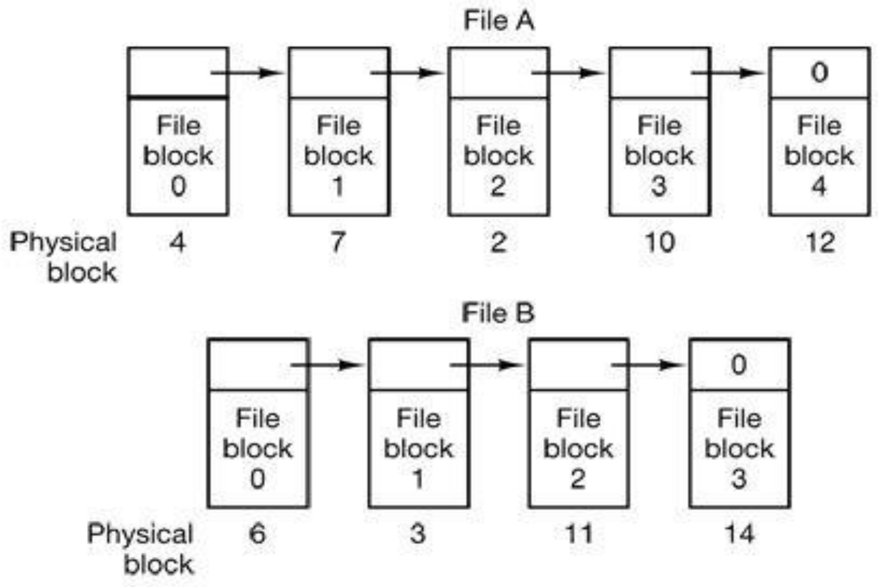


Fig 1: storing a file as a linked list of disk blocks

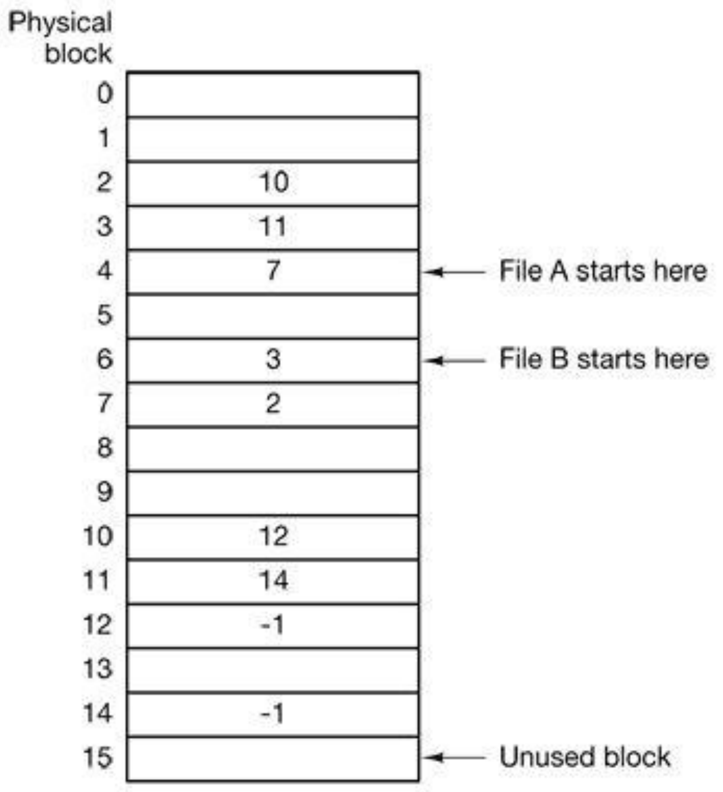


Fig 2: Linked list allocation using a file allocation table in main memo

Both disadvantages of the linked list allocation can be eliminated by taking the pointer word from each disk block and putting it in a table in memory. Figure 2 shows what the table looks like for the example of Fig. 1. In both figures, we have two files. File A uses disk blocks 4, 7, 2, 10, and 12, in that order, and file B uses disk blocks 6, 3, 11, and 14, in that order. Using the table of Fig. 2, we can start with block 4 and follow the chain all the way to the end. The same can be done starting with block 6. Both chains are terminated with a special marker (e.g., –1) that is not a valid block number. Such a table in main memory is called a FAT (File Allocation Table).

MERITS: Using this organization, the entire block is available for data. Furthermore, random access is much easier. Although the chain must still be followed to find a given offset within the file, the chain is entirely in memory, so it can be followed without making any disk references. Like the previous method, it is sufficient for the directory entry to keep a single integer (the starting block number) and still be able to locate all the blocks, no matter how large the file is.

DEMERITS: The primary disadvantage of this method is that the entire table must be in memory all the time to make it work. With a 20-GB disk and a 1-KB block size, the table needs 20 million entries, one for each of the 20 million disk blocks. Each entry has to be a minimum of 3 bytes. For speed in lookup, they should be 4 bytes. Thus the table will take up 60 MB or 80 MB of main memory all the time, depending on whether the system is optimized for space or time.

Conceivably the table could be put in pageable memory, but it would still occupy a great deal of virtual memory and disk space as well as generating extra paging traffic.

Numerical

Here, Total disk size = 200 GB

Block size = 1 KB

$\therefore$ Total number of block = $\frac{200\ GB}{1\ KB} = 200 \times 1024 \times 1024\ KB = 209715200\ KB$

$\therefore$ The FAT table contains 209715200 entries.

Here, size of 1 entry = 3 byte

Total size of FAT = $209715200 \times 3 = 629145600\ bytes = 614400\ KB = 600\ MB$

2(OR). Suppose that a disk has 100 cylinders, numbered 0 to 99. The drive is currently serving a request at cylinder 43, and previous request was at cylinder 25. The queue of pending request, in FIFO order is: 86, 70, 13, 74, 48, 9, 22, 50, and 30.

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all pending request for each of the following disk scheduling algorithms?

a) FCFS

b) SCAN

FCFS:

Total distance (in cylinders) that the disk arm moves to satisfy all pending request

$= (86 - 43) + (86 - 70) + (70 - 13) + (74 - 13) + (74 - 48) + (48 - 9) + (22 - 9) + (50 - 22) + (50 - 30)$

$= 43 + 16 + 57 + 61 + 26 + 39 + 13 + 28 + 20$

$= 303$

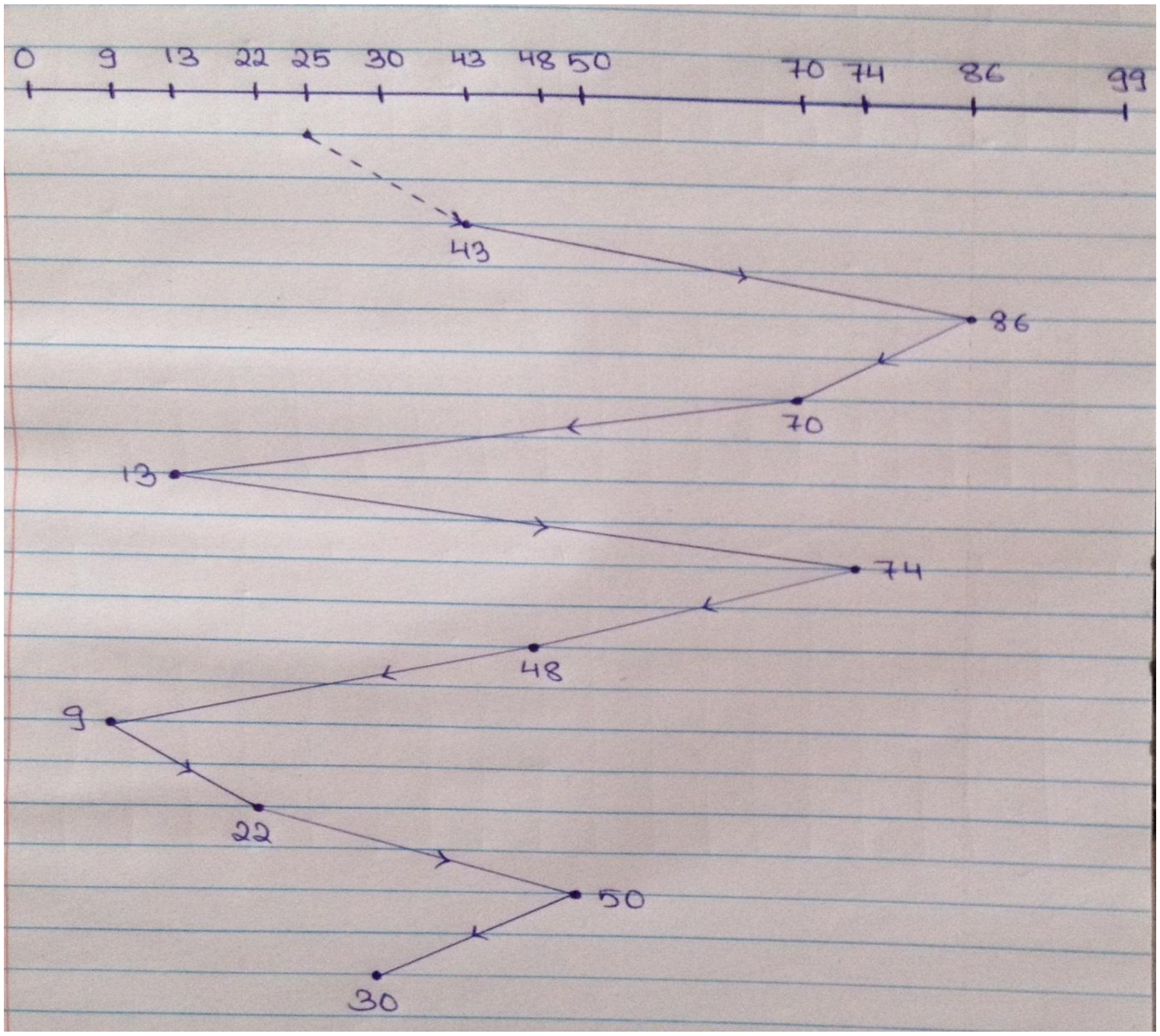


Fig: FCFS

SCAN:

Total distance (in cylinders) that the disk arm moves to satisfy all pending request

$= (48-43)+(50-48)+(70-50)+(74-70)+(86-74)+(99-86)+(99-30)$
$+(30-22)+(22-13)+(13-9)$

$= 5+2+20+4+12+13+69+8+9+4$

$= 146$

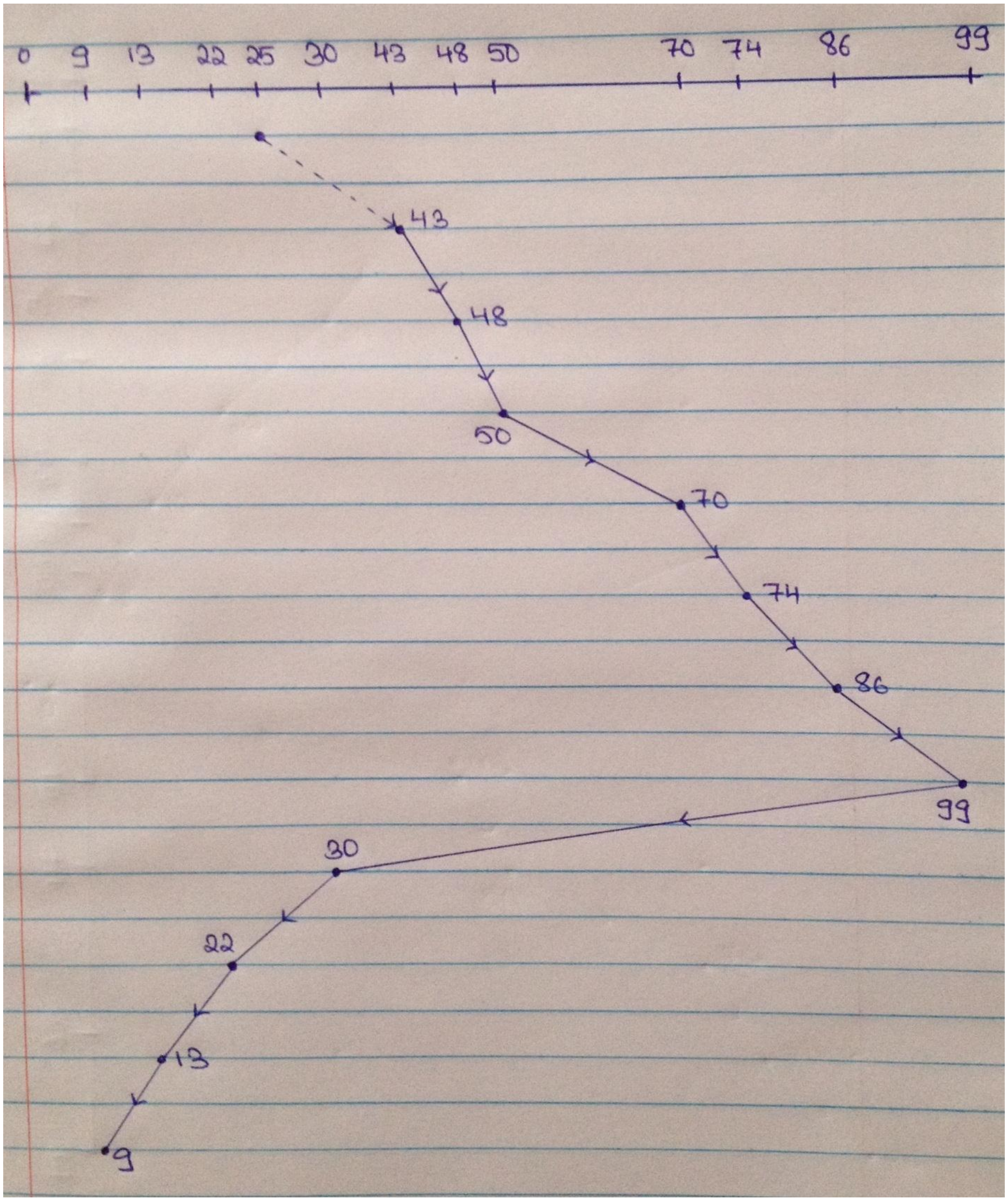


Fig: SCAN

3. Write short notes on:

a) Least recently used page replacement algorithm

b) Segmentation

c) Associative memory

Least recently used page replacement algorithm:

In order to allow the operating system to collect useful statistics about which pages are being used and which ones are not, most computers with virtual memory have two status bits associated with each page. R is set whenever the page is referenced (read or written). M is set when the page is written to (i.e., modified). It is important to realize that these bits must be updated on every memory reference, so it is essential that they be set by the hardware. Once a bit has been set to 1, it stays 1 until the operating system resets it to 0 in software. The R and M bits can be used to build a simple paging algorithm as follows. When a process is started up, both page bits for all its pages are set to 0 by the operating system. Periodically (e.g., on each clock interrupt), the R bit is cleared, to distinguish pages that have not been referenced recently from those that have been. The Least Recently Used algorithm removes a page at random from the lowest numbered nonempty class. Implicit in this algorithm is that it is better to remove a modified page that has not been referenced in at least one dock tick (typically 20 msec) than a clean page that is in heavy use. The main attraction of LRU is that it is easy to understand, moderately efficient to implement, and gives a performance that, while certainly not optimal, may be adequate.

Segmentation:

A straightforward and extremely general solution for one dimensional virtual memory is to provide the machine with many completely independent address spaces, called segments. Each segment consists of a linear sequence of addresses, from 0 to some maximum. The length of each segment may be anything from 0 to the maximum allowed. Different segments may, and usually do, have different lengths. Moreover, segment lengths may change during execution. The length of a stack segment may be increased whenever something is pushed onto the stack and decreased whenever something is popped off the stack. Because each segment constitutes a separate address space, different segments can grow or shrink independently, without affecting each other. If a stack in a certain segment needs more address space to grow, it can have it, because there is nothing else in its address space to bump into. Of course, a segment can fill up but segments are usually very large, so this occurrence is rare. To specify an address in this segmented or two-dimensional memory, the program must supply a two-part address, a segment number, and an address within the segment.

Associative memory:

In most paging schemes, the page tables are kept in memory, due to their large size. Potentially, this design has an enormous impact on performance. Consider, for example, an instruction that copies one register to another. In the absence of paging, this instruction makes only one memory reference, to fetch the instruction. With paging, additional memory references will be needed to access the page table. Since execution speed is generally limited by the rate the CPU can get instructions and data out of the memory, having to make two page table references per memory reference reduces performance by 2/3. Under these conditions, no one would use it. The solution that has been devised is to equip computers with a small hardware device for mapping virtual addresses to physical addresses without going through the page table. The device is called an associative memory or sometimes a TLB (Translation Lookaside Buffer). It is usually inside the MMU and consists of a small number of entries but rarely more than 64. Each entry contains information about one page, including the virtual page number, a bit that is set when the page is modified, the protection code (read/write/execute permissions), and the physical page frame in which the page is located. These fields have a one-to-one correspondence with the fields in the page table. Another bit indicates whether the entry is valid (i.e., in use) or not.

Section B $(8 \times 5 = 40)$

4. What is an operating system? Differentiate between time sharing and real time operating system.

A modern computer system consists of one or more processors, some main memory, disks, printers, a keyboard, a display, network interfaces, and other input/output devices. All in all, a complex system. Writing programs that keep track of all these components and use them correctly, let alone optimally, is an extremely difficult job. For this reason, computers are equipped with a layer of software called the operating system**,** whose job is to manage all these devices and provide user programs with a simpler interface to the hardware.

In real time system, a job has to be completed within fixed deadline (time allowed). If job is not completed within the given time then system may extend time for doing the operations. In time sharing system, fixed time is given to each process and all the processes are arranged in a queue. If the job is not completed within the given time then it jumps to the next job leaving the previous job unfinished. After processing to each job, it again give the same time for unfinished job.

5. Why thread is necessary? In which circumstances user-level thread is better than kernel level thread?

Thread is necessary due to the following reasons:

- Will by default share memory
- Will share file descriptors
- Will share file system context
- Will share signal handling

There are two distinct models of thread controls, and they are *user-level threads* and *kernel-level threads*. The thread function library to implement user-level threads usually runs on top of the system in user mode. Thus, these threads within a process are invisible to the operating system. User-level threads have extremely low overhead, and can achieve high performance in computation. However, using the blocking system calls like read(), the *entire* process would block. Also, the scheduling control by the thread runtime system may cause some threads to gain exclusive access to the CPU and prevent other threads from obtaining the CPU. Finally, access to multiple processors is not guaranteed since the operating system is not aware of existence of these types of threads. On the other hand, kernel-level threads will guarantee multiple processor access but the computing performance is lower than user-level threads due to load on the system. The synchronization and sharing resources among threads are still less expensive than multiple-process model, but more expensive than user-level threads. Thus, user-level thread is better than kernel level thread.

6. Explain about hierarchical directory system with diagrammatic examples.

The two-level hierarchy eliminates name conflicts among users but is not satisfactory for users with a large number of files. Even on a single-user personal computer, it is inconvenient. It is quite common for users to want to group their files together in logical ways. A professor for example, might have a collection of files that together form a book that he is writing for one course, a second collection of files containing student programs submitted for another course, a third group of files containing the code of an advanced compiler-writing system he is building, a fourth group of files containing grant proposals, as well as other files for electronic mail, minutes of meetings, papers he is writing, games, and so on. Some way is needed to group these files together in flexible ways chosen by the user. What is needed is a general hierarchy (i.e., a tree of directories). With this approach, each user can have as many directories as are needed so that files can be grouped together in natural ways. This approach is shown in figure below. Here, the directories *A*, *B*, and *C* contained in the root directory each belong to a different user, two of whom have created subdirectories for projects they are working on.

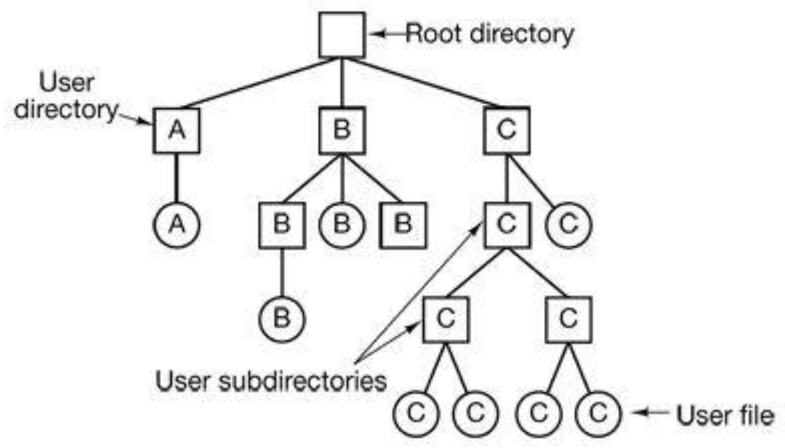


Fig: A hierarchical directory system

7. How can you define the term process scheduling? Differentiate between I/O bound process and CPU bound process.

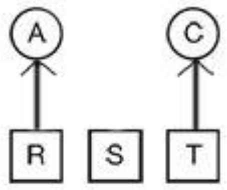
The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.
2. The current process terminates.

3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.
4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the *ready* state. The scheduler may then decide to move this *ready* process into the *running* state.

A program is CPU bound if it would go faster if the CPU were faster, i.e. it spends the majority of its time simply using the CPU (doing calculations). A program that computes new digits of π will typically be CPU-bound, it's just crunching numbers. A program is I/O bound if it would go faster if the I/O subsystem was faster. A program that looks through a huge file for some data will often be I/O bound, since the bottleneck is then the reading of the data from disk.

8. A system has two process and 3 resources. Each process needs a maximum of two resources, is deadlock possible? Explain with answer.



No, deadlock is not possible. If each process is allocated one resource, there is still third resource available, and whichever process takes it, that process will be able to run to completion.

9. What do you mean by interrupt? Explain the working mechanism of interrupt controller.

The hardware mechanism or process that enables a device to notify the CPU is called interrupt. An interrupt is generally initiated by an I/O device, and causes the CPU to stop what it is doing, save its context, jump to the appropriate interrupt service routine, complete it, restore the context, and continue execution.

Interrupts are very important in operating systems, so let us examine the idea more closely. In below figure, we see a three-step process for I/O. In step 1, the driver tells the controller what to do by writing into its device registers. The controller then starts the device. When the controller has finished reading or writing the number of bytes it has been told to transfer, it signals the interrupt controller chip using certain bus lines in step 2. If the interrupt controller is prepared to accept the interrupt (which it may not be if it is busy with a higher priority one), it asserts a pin on the CPU chip informing it, in step 3. In step 4, the interrupt controller puts the number of the device on the bus so the CPU can read it and know which device has just finished (many devices may be running at the same time).

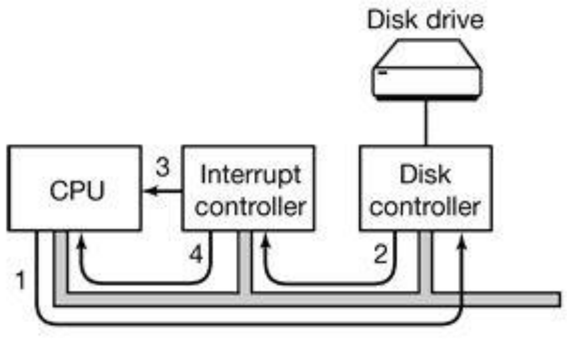


Fig: The steps in starting an I/O device and getting an interrupt

10. Define the term indefinite postponement. How does it differ from deadlock?

Indefinite postponement is also called indefinite blocking or starvation which occurs due to biases in a system's resource scheduling policies. A situation in which all the programs continue to run indefinitely nut fail to make any progress is called starvation. When a process holding resources is denied a request for additional resources, that process must release its held resources and if necessary, request them again together with additional resources. When process releases resources the process may lose its entire works to that point. In such case indefinite postponement or starvation may occur. Aging is the technique that prevents indefinite postponement by increasing Process's priority as it waits for resource.

In the same way deadlock is a situation where a group of processes is permanently blocked as a result of each process having acquired a set of resources needed for its completion and having to wait for release of the remaining resources held by others thus making it impossible for any of the deadlocked processes to proceed. Deadlocks can occur in concurrent environments as a result of the uncontrolled granting of the system resources to the requesting processes. Four conditions for deadlock are:

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

Deadlock can be prevented by denying these four conditions.

11. Explain the mapping of virtual address to real address under segmentation. Compare the throughput (overall performance) of SCAN with SSTF.

OS divides physical memory into partitions. Different partitions can have different sizes.

- Each partition can be given to a process as virtual address space.
- Properties:
    - Virtual address = physical address

- Changing the partition a program is loaded into requires recompilation or relocation (if the compiler produces locatable code)

- Number of processes is limited by the number of partitions size of virtual address space is limited by the size of the partition.

In some systems, a virtual address space can consist of several independent segments. A logical address then consists of two parts: Segment ID and address within segment
Each segment:

- Can grow or shrink independently of the other segments in the same address space
- Has its own memory protection attributes.

A process may have separate segments for code, data, and stack.

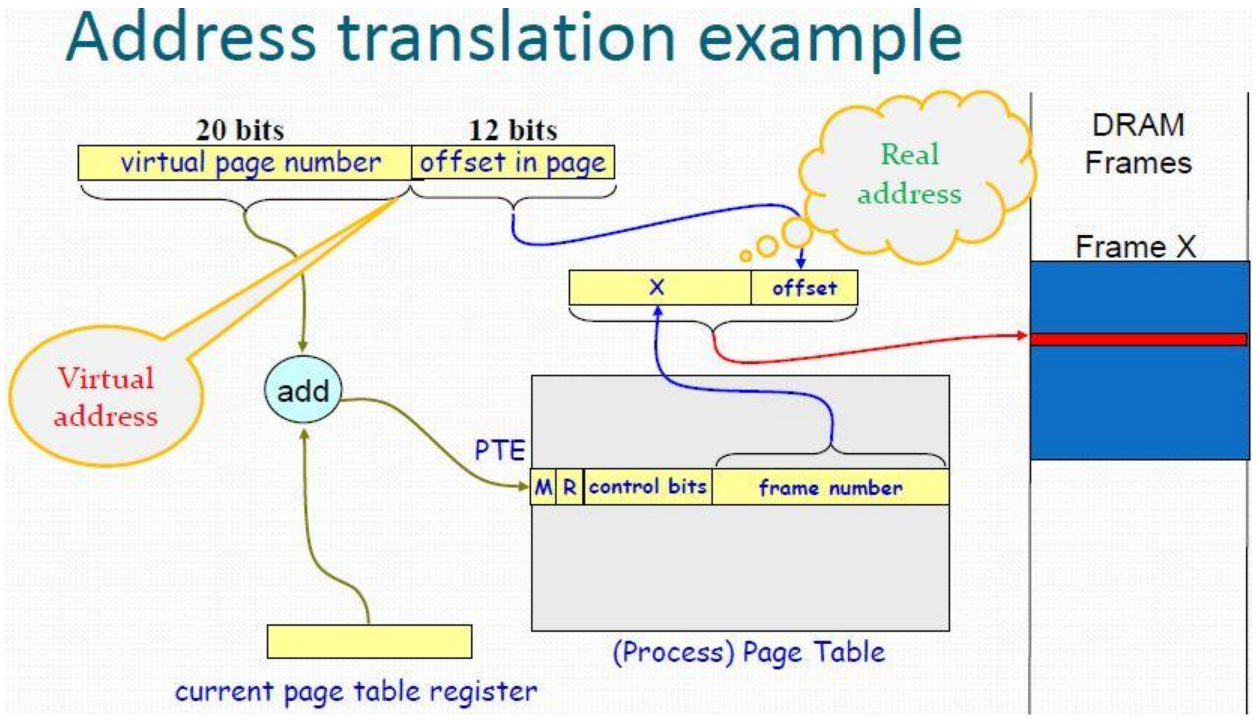
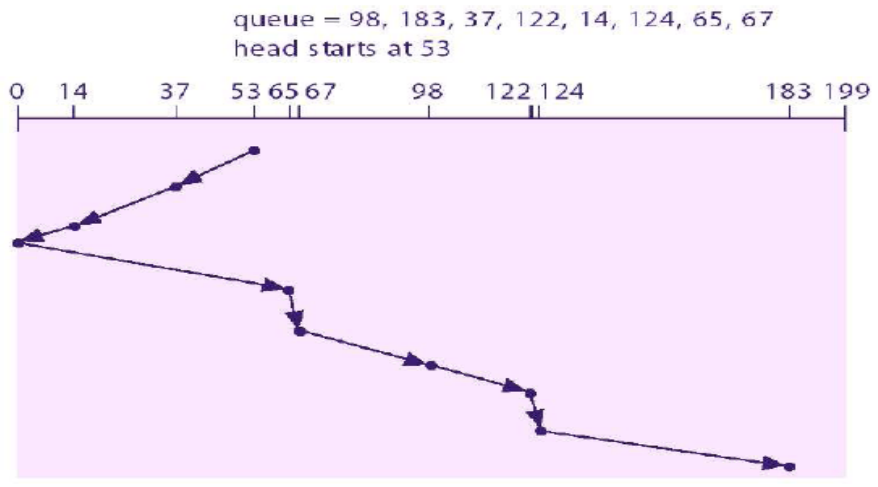


Fig: Mapping of virtual address to real address under segmentation

The throughput performance of SCAN is given below:

- Decreases variances in seek
- Improves response time

SCAN

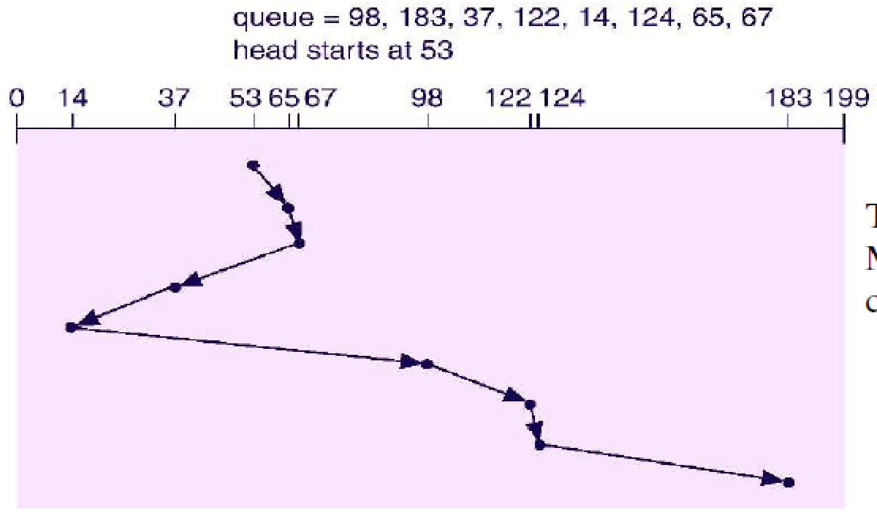


Total Head Movement = 208 cylinders

The throughput performance of SSTF is given below:

- Used in batch system where jobs are maximized per hour
- This gives substantial improvement in performance

Selects the request with the minimum seek time from the current head position.



Total Head Movement = 236 cylinders