Lab. Assignment - 03

Title : Process Control System Calls.

Aim : The demonstration of FORK and WAIT system calls along with zombie and orphan states.

1. Implement C program in which main program accepts the integers to be sorted.
2. Implement C program in which main program accepts an integer array.
3.

Theory :

Process in UNIX -

A process is the basic active entity

Process IDs -

Each process in a linux is identified by its unique process ID. 16 bit numbers that are assigned sequentially by linux.

Creating Processes -

Two common techniques - :

① using system () function.
② using fork () function.

① Using system

The system function in the standard C library provides an easy way to execute a command from within a program, much as if the command had been typed into a shell.

② Using fork.
A process can create a new process by calling fork. The calling process becomes the parent and the created process is called the child. The fork function copies the parent's memory image so that the new process receives a copy of the address space of the parent. Both processes continue at the instruction after the fork statement.

Synopsis :

```
# include <unistd.h>
pid-t fork(void):
```

The fork function returns 0 to the child and returns the child's process ID to the parent.
When fork fails, it returns -1

The wait function - when a process creates a child, both parent and child proceed with execution from the point to the fork. The parent can execute wait to block until the child finishes.

Synopsis:

```
# include <sys/wait.h>
pid_t wait(int * status):
```

If wait returns because the status of a child is reported, these functions return the process ID of that child. If an error occurs, they return -1

Status values -
The status argument of wait is a pointer to an integer variable. If it is not NULL, this function stores the return status of the child in this location.
A zero return value indicates EXIT_SUCCESS any other value indicates EXIT_FAILURE.
POSIX specifies six macros for testing the child's return status. Each takes the status values returned by a child to wait as a parameter.

Synopsis :
```
#include <sys/wait.h>
WIFEXITED (int stat-val)
WEXITSTATUS (int stat-val)
```

exec() system call -
It is used after a fork() system call by one of the two processes to replace the memory space with a new program. The exec() system call loads a binary file into memory and go their separate ways. Within exec family there are functions :
① execl() and execlp()
execl() - It permits us to pass a list of command line arguments to the program.
eg execl("/bin/ls", "ls", "-l", NULL);
execlp() - It does same job except that it will use environmental variable PATH to determine which executable to process.

⑧ execv () and execvp()
execv() - same job as execl () except that command line arguments can be passed to it in the form of an array of pointers to string.

execvp() - it uses environment variable path.

⑨ execve():
int execve (const char* filename, char* const argv[],
 char* const envp []);
It executes the program pointed to by filename. filename must be either a binary executable or a script starting with a line of the form: argv is an array of argument strings.

Process termination -
Normally, a process terminates in one of two ways Either ~~to~~ the executing program calls the exit () function or the program's main function returns.
Each process has an exit code : a number that the process returns to its parent.

Zombie Processes -
If a child process terminates while its parent is calling a wait function, the child process vanishes and its termination status is passed to its parent via the wait call. But what happens when a child process terminates and the parent is not calling wait? Does it simply vanish? No, because then information about its

termination – such as whether it exited normally and if so what its exit status is – would be lost. Instead, when a child process terminates, it becomes a zombie process.

Orphan Processes –.
An orphan process is nearly by the same thing which we see in real world. ~~Or~~ Orphan means someone whose parents are dead. The same way this is a process, whose parents are dead, that means parents are either terminated, killed or exited but the child process is still alive.

Daemon Processes –.
It is a process that runs in the background rather than under the direct control of the user; they are usually initiated as a background processes.

Lab Assignment - 04

Title: SJF & Round Robin CPU sheduling

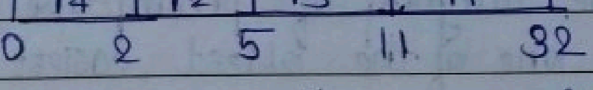
Aim: Implement C program for CPU scheduling algorithms: SJF and Round Robin with different ~~ant~~ arrival time.

Theory:
Shortest Job first scheduling works on the process with the shortest burst time or duration first.
It is of two types-
① Non Preemptive
② Preemptive.

Consider below processes-

Process	Burst Time.
P1	21
P2	3
P3	6
P4	2

In SJF, shortest process is executed first
Gantt chart:



As we can see, the process P4 will be picked up first as it has the shortest burst time, then P2 followed by P3 and at last P1.
~~We schedule the same set of processes using~~ FCFS

Preemptive SJF:
Jobs are put into ready queue as they arrive but as a process with short ~~bus~~ burst time arrives the existing process is preempted or removed from execution and shorter job is executed first.
As you can see in the Gantt chart,
P1 arrives first, hence its execution starts immediately but just after 1ms, process P2 arrives with a burst time of 3ms which is less than the burst time of P1. hence P1 is ~~pee~~ preempted and process P2 is executed. As P2 is getting executed after 1ms, P3 arrives but it has a burst time greater than P2 hence P2 continues to execute. But after another millisecond P4 arrives with burst time 2ms as a result P2 is preempted and P4 is executed. After completion of P4, P2 is picked up and finishes. P2 will get executed and at last P1.

Round Robin scheduling algorithm is mainly designed for time sharing systems. This algorithm is similar to FCFS, but in RR preemption is added which enables the system to switch between processes. A fixed time is alloted, called a quantum for execution.
Characteristics of RR:
① Resides under the category of preemptive algorithms
② This algorithm is one of the oldest, easiest and fairest
③ It is realtime because it responds to the event within a specific time limit.
④ It is hybrid model and clock driven in nature

Impordant terms —

① Completion time: It is the time at which any process completes its execution.

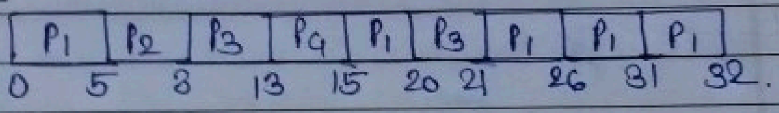
② Turn Around time: This mainly indicates, the time difference between completion and arrival time.

③ Waiting time: It indicates the time difference between turn around time and burst time.

example

Process.	BT	TAT	WT.
P1	21	32	11
P2	3	8	5
P3	6	21	15
P4	2	15	13.

Gantt chart—



Advantages.

① A particular time quantum is allocated to different jobs.

② It is cyclic nature.

③ newly created process is added to end of ready queue.

④ employs time sharing

⑤ Spends more time on context switching.

⑥ offers larger waiting and response time.

Lab Assignment - 05

Title: Thread synchronisation using counting semaphores.

Aim: Application to demonstrate producer-consumer problem with counting semaphores and mutex.

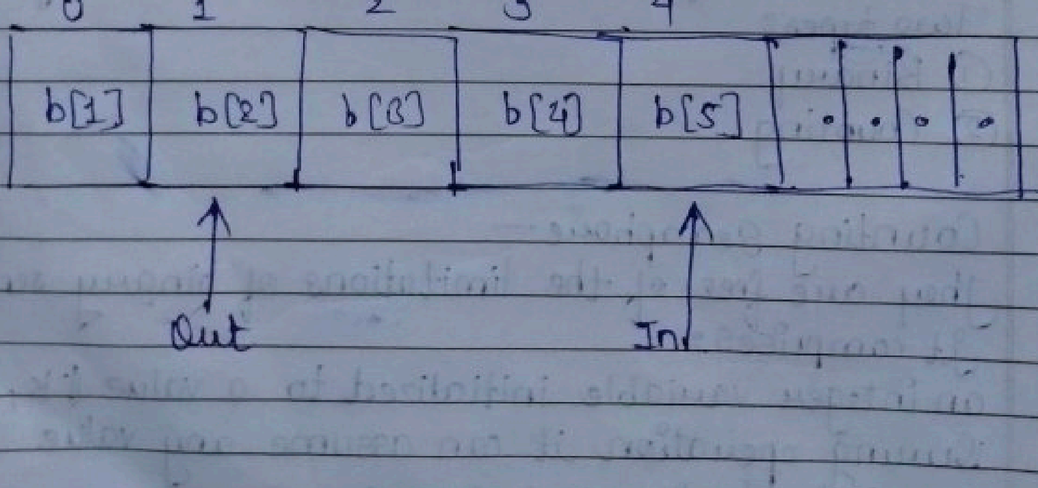
Theory:
Semaphores -
An integer value used for signaling among processes
Only three operations may be performed
on a semaphore, all of which are atomic:
initialize, decrement and increment
The decrement operation may result in blocking
the process ~~on in~~ and increment operation
in unblocking the process called counting semaphores.
Two types:
① Binary
② Counting.

Counting semaphore -
They are free of the limitations of binary semaphores
It comprises:
an integer variable initialized to a value (k, k>=0).
During operation it can assume any value <= k
a pointer to process a queue.

NMIET

The producer / consumer problem –
One of the most common problem faced in concurrent processing.
The general statement is:
there are one or more producers generating some type of data and placing these in a buffer. There is a single consumer that is taking items out of a buffer one at a time. The system is to be constrained to prevent the overlap of buffer operations. That is only one agent may access the buffer at any one time.
It is to make sure producer won't try to add data into the buffer if its full.

Let us assume that buffer is infinite and consists of a linear array of elements. Each time an index into the buffer is incremented, the ~~customer~~ consumer proceeds



Lab Assignment - 06.

Title: Thread synchronization and mutual exclusion using mutex.

Aim: Application to demonstrate Reader Writer problem with reader priority.

Theory:

Semaphores -
An integer value used for signaling among processes.
Only three operations may be performed on a semaphore; all of which are atomic - initialize, decrement and increment.
The decrement operation may result in the blocking of a process, and the increment in unblocking the process.
Two types: ① counting ② Binary.

Operation of a counting semaphore -
1. Let the initial value of semaphore count be 1.
2. When semaphore count = 1, it implies that no process is executing in its critical section and no process is waiting in queue.
3. When semaphore count = 0, it implies that one process is executing in its critical section but no process is waiting in queue.
4. When semaphore count = N, it implies that one process is executing in critical section and N processes are

NMIET

waiting.
5. When a process is waiting, it is not performing any busy waiting. It is rather in "waiting" or "blocked" state
6. When a process is waiting in queue is selected for entry into its critical section, it is transferred to "ready" state.

Reader-Writer problem with readers for priority.
It is defined as —
There is data area shared among a number of person processes. The data area could be a file, a block of main memory, or even a ~~block~~ bank of registers. There are number of processes that only read the data area and a number that only write to the data area. The conditions that must be satisfied are —
① Any number of readers may simultaneously read the file.
② Only one writer at a time may write to a file
③ If a writer is writing to file, no reader may ~~reader read real~~ read it.

Threads —
Multiple strands of execution in a single program are called threads. A more precise definition is that a thread is a sequence of control within a process. like many other OS, linux is quite capable of running multiple processes simultaneously. Indeed, all processes have at least one thread of execution.

POSIX thread in ~~Linux~~ Unix -
Including the file pthread.h provides us with other definitions and prototypes that we will need in our code, much like stdio.h for standard input and output routines.

```
#include <pthread.h>
int pthread_create (pthread_t * ~~trea~~ thread. pthread_attr_
                     t* attr,)
 void * (*start_routine) (void*), void* arg ;
```

This function is used to create thread.

Linux Semaphore facilities (Binary Semaphore) -.
A semaphore is created with the sem_init function. It initializes a semaphore object pointed to by sem, sets its sharing option and gives it an integer value.

Lab Assignment - 07

Title : Banker's Algorithm

Aim : Implement C program for deadlock avoidance

Theory :
Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking system to determine whether a loan can be granted or not.
Consider there are n account holders in a bank and the sum of money in all their accounts is S. Everytime a loan has to be granted by the bank, it subtracts the loan amount from the total money bank has. The it checks if that difference is greater than S. It is done because only then, the bank would have enough money even if all the n account holders draw all their money at once.

Characteristics :-
① If any process requests for a resource, then it has to wait
② Consists of advanced features for maximum resource collection
③ If any processes use all the resources, then it has to return them in restricted time period.

Some data structures used to implement banker's algorithm are -

1. Available: It is an array of length m. It represents the number of available resources of each type. If Available [j] = k, then there are k instances available of resource type Rj.

2. Max: It is an nxm matrix which represents the maximum number of instances of each resource than a process can request. If Max[i][j] = k, then the processes Pi can request almost k instances of resource type Rj.

3. Allocation: It is an nxm matrix which represents the number of resources of each type currently allocated to each process.

4. Need: It is a two dimensional array. It is an nxm matrix which indicates the remaining resource needed for each process.

Need [i][j] = Max [i][j] - Allocation [i][j].

~~It comprises~~

Banker's Algorithm comprises of two algorithms -
① Safety algorithm
② Resource request algorithm.

(A) Safety algorithm
It is an algorithm used to find whether or not a system is in its safe state
<1> Let Work and finish be vectors of length m and n respectively. Initially,
Work = Available
finish [i] = false for i = 0, 1, . . n-1
<ii> find an index i such that both
finish [i] == false
Need[i] <= Work.
<iii> Perform
Work = Work + Allocation
finish [i] = true
Goto <ii>
<iv> If finish == true for all i, then system is in safe state.

(B) Resource Request Algorithm
It is mainly used to determine whether requests can be safely granted or not.
<i> If Request [i] <= Need [i] then goto <ii>
else raise an error condition.
<ii> If Request [i] <= Available [i] then goto <iii>
else Pi must ~~not~~ have to wait.
<iii> Available = ~~Ava~~ Available - Request
Allocation = Allocation + Request
Need = Need ~~*~~ - Request.

Disadvantages—.

① During the time of processing, this algorithm does not permit a process to change its maximum need.

② All processes must know in advance about maximum need of resources.

③ permits the requests to be provided in constrained time, but for one year which is a fixed period.

Lab Assignment - 08.

Title: Page Replacement Algorithms.

Aim: Implement the C program for Page Replacement Algorithms in unix / Linux and how to implement in C

Theory:
What are page Replacement Algorithms?
As studied in demand paging, only certain pages of a process are loaded initially into the memory. This allows us to get more pages and no free memory is available to bring them in.
following steps can be taken—
1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Remove some other process completely from the memory to free frames.
3. find some pages that are not being used right now, move them to the disk to get free frames. This technique is called page replacement algorithm.
4. In this case, if a process requests a new page and supposes there are no free frames, then the OS needs to decide which page ~~and suppose~~ to replace. The operating system must use any page replacement algorithm in order to select the victim frame. The operating system must ~~use~~ then write the victim frame to ~~discuss~~ disk then read the desired page into the frame and then update the page tables.

1. Page replacement ~~provide~~ prevents the over allocation of the memory by modifying the page fault service routine.

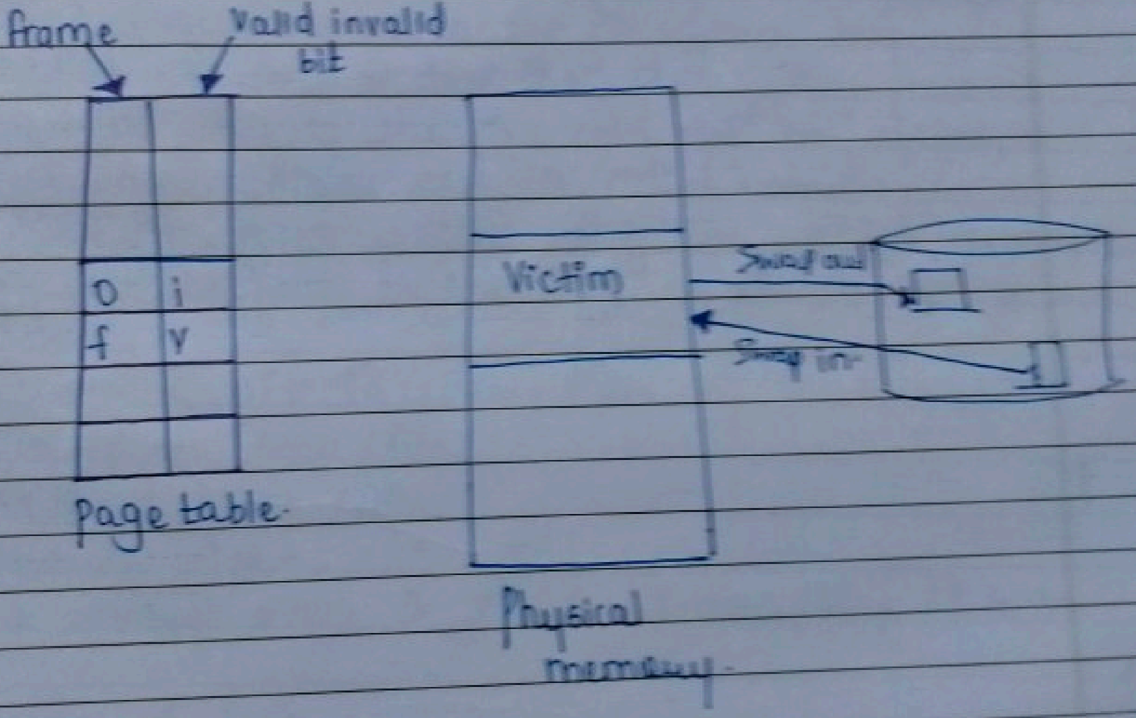
2. To reduce the overhead of page replacement a modify bit is used in order to indicate whether each page is modified.

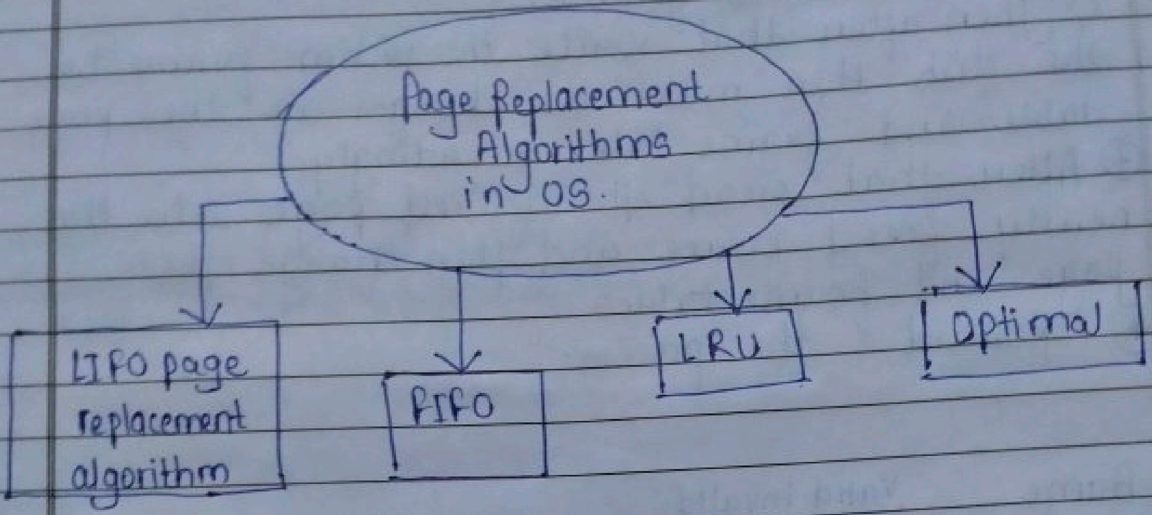
3. This technique provides complete separation between logical memory and physical memory.

Page Replacement in OS —.
In virtual Memory Management, Page Replacement Algorithms plays an important role.
The main objective of all the policies is to decrease the maximum number of page faults.
Page fault : It is basically a memory error and it occurs when the current program's attempt to access the memory page for mapping into virtual address space, but it is unable to load into physical memory then this is refferred to as page fault.

Basic algorithm —
① Find the location of the desired page on the disk.
② Find a free frame —
a) If there is a free frame, then use it
b) If there is no free frame then make use of the page replacement algorithm in order to

select the victim frame.
c) Then after that write the victim frame to the disk then make the changes in the page table and frame table accordingly.
③ After that read the desired page into the newly freed frame and then change the page and frame tables.
④ Restart the process.




Page Replacement Algorithms in OS.
LIFO page replacement algorithm
FIFO
LRU
Optimal

Lab Assignment : 09

Title: Inter process Communication

Aim: Inter process communication in Linux
using FIFOs.

Theory:
FIFOs —
A first in first out (FIFO) file is a pipe that has a name in the filesystem. Any process can open or close the FIFO : the processes on either end of the pipe need not be related to each other. FIFOs are also called named pipes.
You can make a FIFO using the mkfifo command. Specify the path to the FIFO on the command line.
for eg:
create a FIFO in /tmp/fifo by invoking —

```
% mkfifo /tmp/fifo
% ls - l / tmp /fifo
prw-rw-rw-
 1 samuel users 0 Jan16 14·04 /tmp/fifo
```

Creating a FIFO —
Create a FIFO programmatically using the mkfifo function. The first argument is the path at which to create the FIFO ; the second parameter specifies the pipe's owner, group and world permissions.

Accessing a fifo :-
Access a fifo just like an ordinary file. To communicate through fifo, one program must open it for writing and another program must open it for reading either I/O functions may be used.

Lab Assignment -10

Title: Inter process communication using shared memory using system V

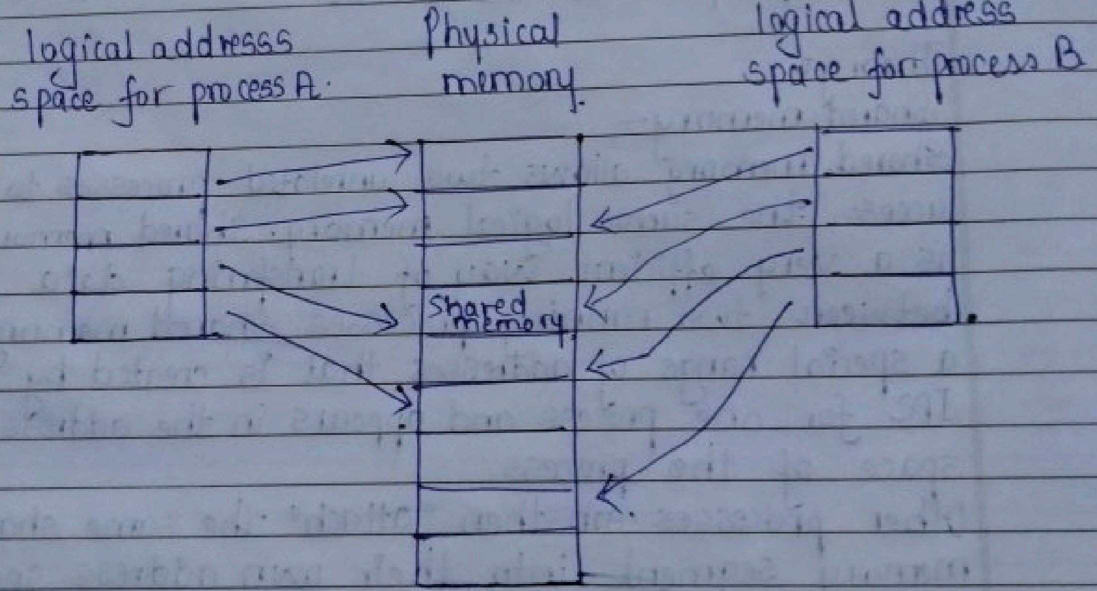
Aim: Application to demonstrate : Client and Server programs in which server process creates a shared memory segment and write the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

Theory:

Shared memory-

Shared memory allows two unrelated processes to access the same logical memory. Shared memory is a very efficient way of transferring data between two running processes. Shared memory is a special range of addresses that is created by IPC for one process and appears in the address space of the process.
Other processes can then "attach" the same shared memory segment into their own address space.
All processes can acess the memory locations just as if the memory had been allocated by malloc. If one process writes to the shared memory, the changes immediately become visible to any other process that has access to the shared memory.
Shared memory provides an efficient way of sharing and passing data between multiple processes.

By itself, shared memory doesn't provide any synchronization facilities. Because it provides no synchronization facilities.
Typically, we use shared memory to provide efficient access to large areas of memory and pass small messages to synchronize access to that memory.
There are no automatic facilities to prevent a second process from starting to read the shared memory befored the first process has finished writing to it.


logical addresss space for process A
Physical memory
logical address space for process B
Shared memory

The arrows show the mapping of the logical address space of each process to the physical memory available.

Lab Assignment - 11.

Title : Disk Scheduling Algorithms

Aim : To implement C program for Disk scheduling algorithms.
① SSTF
② SCAN
③ C-LOOK.

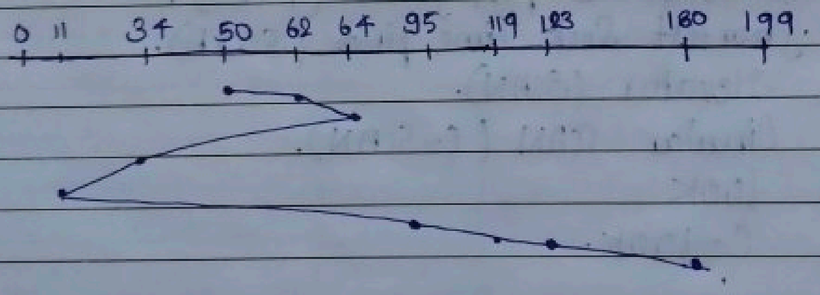
Theory :
In OS, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk scheduling algorithms are used to reduce the total seek time of any request.
Types of disk-scheduling algorithms —
① first come first serve (FCFS).
② Shortest Seek Time first (SSTF).
③ Elevator (SCAN).
④ Cirular SCAN (C-SCAN).
⑤ LOOK
⑥ C-LOOK.

1. first Come first serve (FCFS) —
All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be next number served. Using these algorithms ~~is keep~~ doesn't provide the best results. To determine the number of head

movements you would simply find the number of tracks it took to move from one request to the next.
Given queue- 95, 180, 34, 119, 11, 123, 62, 64.
with read write head initially at the track 50 and tail track being at 199

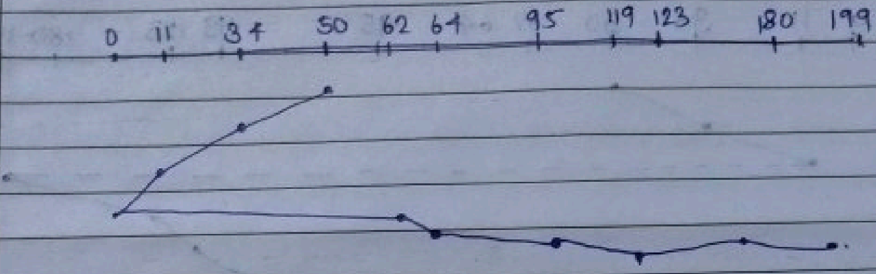
for this case, it went from 50 to 95 to 180 and so on.

2. Shortest Seek Time first (SSTF)
In this case request is serviced according to next shortest distance. Starting at 50 the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. There is a great chance that starvation would take place.

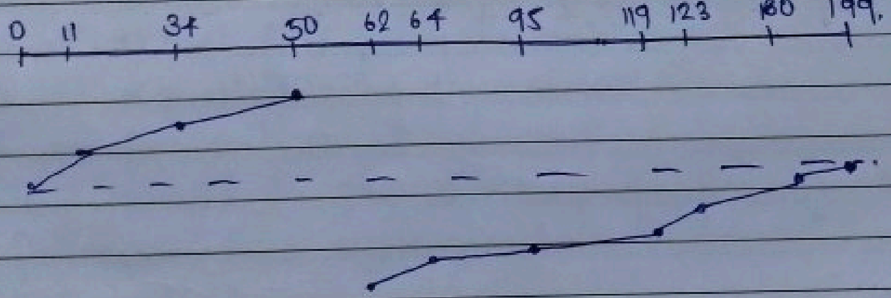


3. Elevator (SCAN).
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a requests comes in after it has been scanned it will not be serviced until

the process comes back down or moves back up.

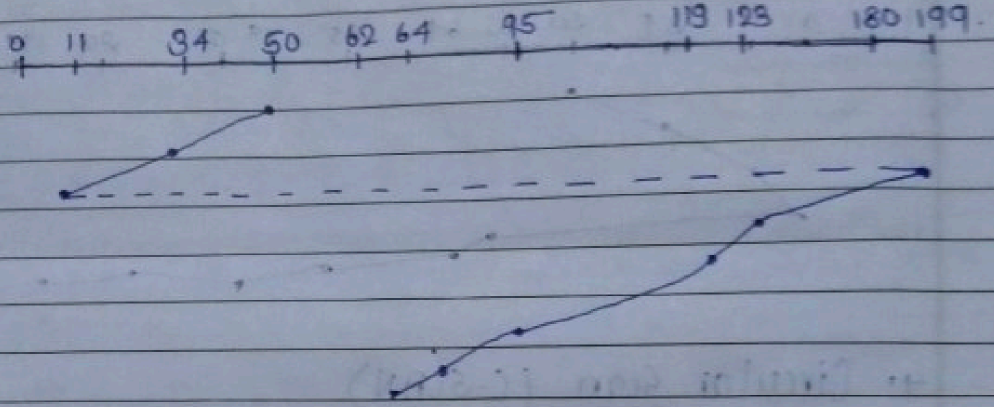


4. Circular Scan (C-SCAN).
Circular scanning works just like elevator to some extent. It begins its scan towards the nearest end and works it way all the way to end of the system. Once it hits the bottom or top it jump to the other end and moves in the same direction.



5. C-LOOK
This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that ~~is a~~ it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request, C-SCAN had a total movement of 187 but

This scan reduced it down to 157 tracks.



9850390031. NMIET

Lab Assignment - 12

Title : System call.

Aim : Implement a new system call, add this new system call in the Linux Kernel and demonstrate the use of same.

Theory :
Adding a simple system call -
1. Download the kernel source :
In your terminal type following command.

```
wget https://www.kernel.org/pub/linux/kernel/
  v4.x/linux-4.17.4.tar.xz
```

2. Extract the kernel source code :

```
sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/
```

tar - stores and extracts files from a tape or disk.
-x - extract files from an archive.
-v - requested using the -verbose option
-f - file archive.
-C - extract to the directory specified after it.
3. Define a new system call
sys-hello()
Create a directory named hello/ and change the directory to hello/ :

```
mkdir hello
  cd hello
```

Create a file hello.c using text editor

```
gedit hello.c
```

create a "Makefile" in the hello directory.

gedit makefile.
and add the following line to it —
obj - y = hello.o
4. Adding hello/ to the kernel's makefile
Go back to the parent dir i.e. cd/ and open "Makefile"
gedit makefile.
5. Add the new system call to system call table.
If you are on 32 bit system you'll need to change 'syscall-32.tbl' for 64 bit, change 'syscall-64.tbl'.
6. Add the new system call to the system call header file:
Go to the linux-4.17.4/ directory and type —
cd include/linux/
gedit syscalls.h
Save and exit. This defines the prototype of function of our system call. "asmlinkage" is a key word used to indicate that all parameters of the function would be available on the stack.
7. Compile the kernel:
Before starting to compile you need to install a few packages. Type —
sudo apt-get install gcc.
sudo apt-get install libncurses5-dev
sudo apt-get install bison.
sudo apt-get install flex.
sudo apt-get install libelf-dev.
sudo apt-get update
sudo apt-get upgrade.

to configure kernel use -
sudo make menuconfig.
Once the above command is used to configure the linux kernel, you will get a pop up window with the list of menus and you can select items for the new configuration
Now to update the kernel in your system reboot the system. You can use~~d~~ following command -
shutdown -r now.
9. Test system call :-
Go to your home (~) directory using -
cd ~
and create a ~~uspa~~ userspace.c file -
gedit userspace.c.

Now compile and run the program.
gcc userspace.c.
./a.out.
If all steps are done correctly, you'll get an output -
system call sys-hello returned 0.
Now, to check the message of your kernel run -
dmesg.
Then display hello world at the end of the kernel!..