

**Ex. No.: 4a)**

**Date:**

### **EMPLOYEE AVERAGE PAY**

**Aim:**

To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4.

**Algorithm:**

1. Create a flat file emp.dat for employees with their name, salary per day and number of days worked and save it.
2. Create an awk script emp.awk
3. For each employee record do
  - a. If Salary is greater than 6000 and number of days worked is more than 4 then print name and salary earned
  - b. Compute total pay of employee
4. Print the total number of employees satisfying the criteria and their average pay.

**Program Code:**

```
//emp.awk
BEGIN{print "EMPLOYEES DETAILS"}
{#salary should be greater than 6000 and days
more than 4
if($2>6000 && $3>4)
{
print $1,"\\t\\t", $2*$3
pay=pay+ $2*$3
count=count+1
}
}
```

```
END{
{#action part
print "no of employees are=",NR/count+1
print "total pay=",pay
print "average pay=",pay/count
}
}
```

```
//emp.dat – Col1 is name, Col2 is Salary Per Day and Col3 is //no. of days worked
JOE 8000 5
RAM 6000 5
```

```
TIM 5000 6
BEN 7000 7
AMY 6500 6
```

**Output:**

```
[student@localhost ~]$ vi emp.dat
[student@localhost ~]$ vi emp.awk
[student@localhost ~]$ gawk -f emp.awk emp.dat.
EMPLOYEES DETAILS
JOE 40000
BEN 49000
AMY 39000
no of employees are= 3
total pay= 128000
average pay= 42666.7
[student@localhost ~]$
```

**Ex. No.: 4b)**

**Date:**

### **RESULTS OF EXAMINATION**

**Aim:**

To print the pass/fail status of a student in a class.

**Algorithm:**

1. Read the data from file
2. Get a data from each column
3. Compare the all subject marks column
  - a. If marks less than 45 then print Fail
  - b. else print Pass

**Program Code:**

**//marks.dat**

//Col1- name, Col 2 to Col7 – marks in various subjects

BEN 40 55 66 77 55 77

TOM 60 67 84 92 90 60

RAM 90 95 84 87 56 70

JIM 60 70 65 78 90 87

**//marks.awk**

```
BEGIN{
    print "NAME", "\t", "SUB-1", "\t", "SUB-2", "\t", "SUB- 3", "\t", "SUB-4", "\t", "SUB
5", "\t", "SUB-6", "\t", "STATUS"
    print"_____ \n" }
{ #BODY
    if ( $2 < 45 || $3 < 45 || $4 < 45 || $5 < 45 || $6 < 45
        || $7 < 45)
    {
        print $1, "\t", $2, "\t", $3, "\t", $4, "\t", $5, "\t",
            $6, "\t", $7, "\t", "FAIL"
    }
    else
    {

        print $1, "\t", $2, "\t", $3, "\t", $4, "\t", $5, "\t",
            $6, "\t", $7, "\t", "PASS"
        }
    }
}
```

```
END {  
    print "_____\\n" }
```

**Output:**

```
[root@localhost student]# gawk -f marks.awk marks.dat
```

```
NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS
```

---

```
BEN 40 55 66 77 55 77 FAIL TOM 60 67 84 92 90 60 PASS RAM 90 95 84 87  
56 70 PASS JIM 60 70 65 78 90 87 PASS
```

---

**Ex. No.: 5**

**Date:**

### **System Calls Programming**

**Aim:** To experiment system calls using `fork()`, `execlp()` and `pid()` functions.

#### **Algorithm:**

**1. Start**

- Include the required header files (`stdio.h` and `stdlib.h`).

**2. Variable Declaration**

- Declare an integer variable `pid` to hold the process ID.

**3. Create a Process**

- Call the `fork()` function to create a new process. Store the return value in the `pid` variable:
  - If `fork()` returns:
    - -1: Forking failed (child process not created).
    - 0: Process is the child process.
    - Positive integer: Process is the parent process.

**4. Print Statement Executed Twice**

- Print the statement:

```
scss
Copy code
THIS LINE EXECUTED TWICE
```

(This line is executed by both parent and child processes after `fork()`).

**5. Check for Process Creation Failure**

- If `pid == -1`:
  - Print:

```
Copy code
CHILD PROCESS NOT CREATED
```
  - Exit the program using `exit(0)`.

**6. Child Process Execution**

- If `pid == 0` (child process):
  - Print:
    - Process ID of the child process using `getpid()`.
    - Parent process ID of the child process using `getppid()`.

**7. Parent Process Execution**

- If `pid > 0` (parent process):
  - Print:
    - Process ID of the parent process using `getpid()`.
    - Parent's parent process ID using `getppid()`.

## 8. Final Print Statement

- Print the statement:

```
objectivec
Copy code
IT CAN BE EXECUTED TWICE
```

(This line is executed by both parent and child processes).

## 9. End

### Program:

#### **FORK SYSTEM CALL PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
int main( )
{
int pid;
pid=fork();
printf("\n THIS LINE EXECUTED TWICE");
if(pid== -1) {
printf("\n CHILD PROCESS NOT CREATED\n");

exit(0);
}
if(pid==0) {
printf("\n I AM CHILD PROCESS AND MY ID IS %d \n",getpid( ));
printf("\n THE CHILD PARENT PROCESS ID IS:%d \n",getppid( ));
}
else
{
printf("\n I AM PARENT PROCESS AND MY ID IS:%d\n",getpid( ));
printf("\n THE PARENTS PARENT PROCESS ID IS:%d\n",getppid( ));
}
printf("\n IT CAN BE EXECUTED TWICE");
printf("\n");
}
```

**Ex. No.: 6a)**

**Date:**

### **FIRST COME FIRST SERVE**

**Aim:**

To implement First-come First- serve(FCFS) scheduling technique

**Algorithm:**

1. Get the number of processes from the user.
  2. Read the process name and burst time.
  3. Calculate the total process time.
  4. Calculate the total waiting time and total turnaround time for each process 5.
- Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

**Program Code:**

```
bt=[]
print("Enter the number of process: ")
n=int(input())
print("Enter the burst time of the processes: \n")
bt=list(map(int, input().split()))
wt=[]
avgwt=0
tat=[]
avgtat=0
wt.insert(0,0)
tat.insert(0,bt[0])
for i in range(1,len(bt)):
    wt.insert(i,wt[i-1]+bt[i-1])
    tat.insert(i,wt[i]+bt[i])
    avgwt+=wt[i]
    avgtat+=tat[i]
avgwt=float(avgwt)/n
avgtat=float(avgtat)/n
print("\n")
print("Process\t Burst Time\t Waiting Time\t Turnaround Time")
for i in range(0,n):
    print(str(i)+"\t\t"+str(bt[i])+"\t\t"+str(wt[i])+"\t\t"+str(tat[i]))
    print("\n")
print("Average Waiting time is: "+str(avgwt))
print("Average Turn Around Time is: "+str(avgtat))
```

**Output:**

Enter the number of process:

3

Enter the burst time of the processes:

24 3 3

Process Burst Time Waiting Time Turn Around Time 0 24 0 24

1 3 24 27

2 3 27 30

Average Waiting time is: 17.0

Average Turn Around Time is: 19.0



**Ex. No.: 6b)**

**Date:**

### **SHORTEST JOB FIRST**

**Aim:**

To implement the Shortest Job First(SJF) scheduling technique

**Algorithm:**

1. Declare the structure and its elements.
2. Get number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero.
5. Sort based on burst time of all processes in ascending order
6. Calculate the waiting time and turnaround time for each process.
7. Calculate the average waiting time and average turnaround time.
8. Display the results.

**Program Code:**

```
bt=[] #bt stands for burst time
print("Enter the number of process: ")
n=int(input())
processes=[]
for i in range(0,n):
    processes.insert(i,i+1)
print("Enter the burst time of the processes: \n")
bt=list(map(int, raw_input().split()))
for i in range(0,len(bt)-1): #applying bubble sort on bt
    for j in range(0,len(bt)-i-1):
        if(bt[j]>bt[j+1]):
            temp=bt[j]
            bt[j]=bt[j+1]
            bt[j+1]=temp
    temp=processes[j]
    processes[j]=processes[j+1]
    processes[j+1]=temp
wt=[] #wt stands for waiting time
avgwt=0 #average of waiting time
tat=[] #tat stands for turnaround time
avgtat=0 #average of total turnaround time
wt.insert(0,0)
tat.insert(0,0)
for i in range(1,len(bt)):
    wt.insert(i,wt[i-1]+bt[i-1])
    tat.insert(i,wt[i]+bt[i])
    avgwt+=wt[i]
    avgtat+=tat[i]
avgwt=float(avgwt)/n
avgtat=float(avgtat)/n
print("\n")
```

```

print("Process\t Burst Time\t Waiting Time\t Turn Around
Time")
for i in range(0,n):
    print(str(processes[i])+"\t"+str(bt[i])+"\t"+str(wt[i])
+" \t"+str(tat[i]))
print("Average Waiting time is: "+str(avgtat))
print("Average Turn Around Time is: "+str(avgtat))

```

### **Output:**

Enter the number of process:

4

Enter the burst time of the processes:

8 4 9 5

Process Burst Time Waiting Time Turn Around Time 2 4 0 4

4 5 4 9

1 8 9 17

3 9 17 26

Average Waiting time is: 7.5

Average Turn Around Time is: 13.0

**Ex. No.: 6c)**

**Date:**

## **PRIORITY SCHEDULING**

**Aim:**

To implement priority scheduling technique

**Algorithm:**

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority 4.
- Calculate the total waiting time and total turnaround time for each process 5.
- Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time

**Program Code:**

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1; //contains process number
    }
    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;

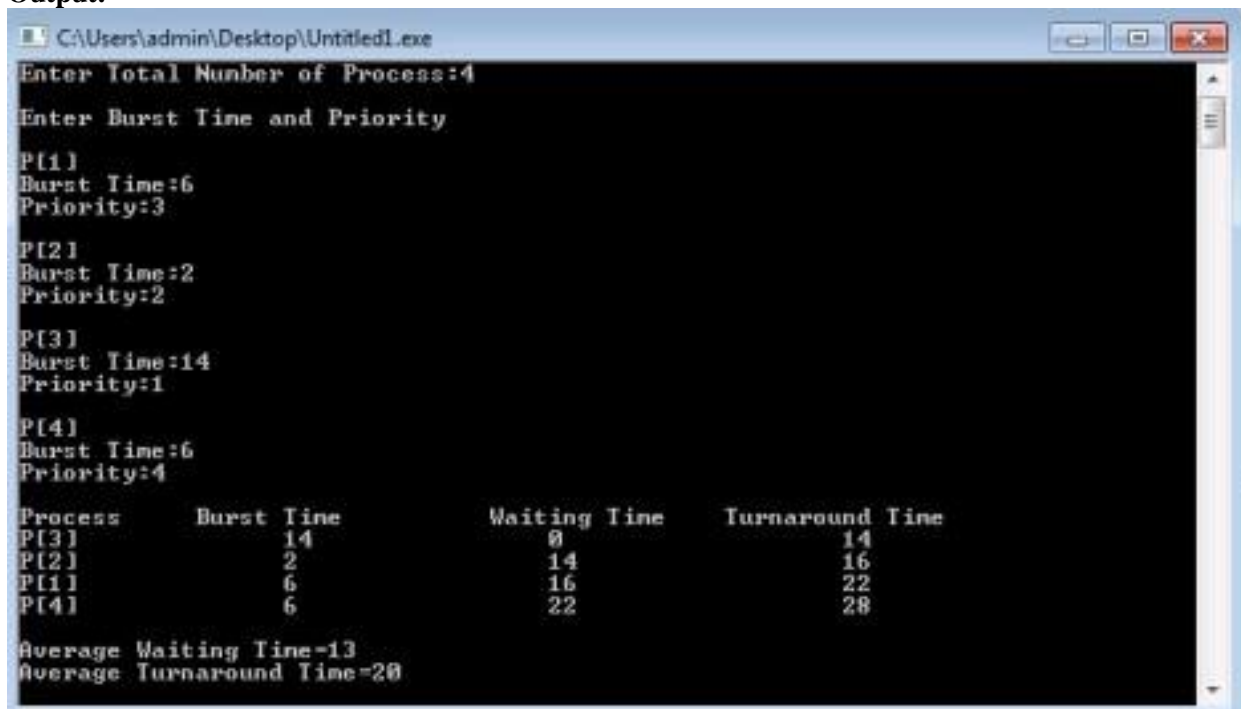
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
}
```

```

p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0; //waiting time for first process is zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=total/n; //average waiting time
total=0;
printf("\nProcess\tBurst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\nP[%d]\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]); }
avg_tat=total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}

```

### Output:



```

C:\Users\admin\Desktop\Untitled1.exe
Enter Total Number of Process:4
Enter Burst Time and Priority
P[1]
Burst Time:6
Priority:3
P[2]
Burst Time:2
Priority:2
P[3]
Burst Time:14
Priority:1
P[4]
Burst Time:6
Priority:4
Process      Burst Time      Waiting Time      Turnaround Time
P[3]          14              0                14
P[2]           2             14               16
P[1]           6             16               22
P[4]           6             22               28
Average Waiting Time=13
Average Turnaround Time=28

```

**Ex. No.: 6d)**

**Date**

### **ROUND ROBIN SCHEDULING**

**Aim:**

To implement the Round Robin (RR) scheduling technique

**Algorithm:**

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem\_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0.
6. Initialize time :  $t = 0$
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
  - a- If  $\text{rem\_bt}[i] > \text{quantum}$ 
    - (i)  $t = t + \text{quantum}$
    - (ii)  $\text{bt\_rem}[i] -= \text{quantum}$ ;
  - b- Else // Last cycle for this process
    - (i)  $t = t + \text{bt\_rem}[i]$ ;
    - (ii)  $\text{wt}[i] = t - \text{bt}[i]$
    - (iii)  $\text{bt\_rem}[i] = 0$ ; // This process is over
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

**Program Code:**

```
#include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10]; float
    average_wait_time, average_turnaround_time;
    printf("\nEnter Total Number of Processes:t");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:t");

        scanf("%d", &arrival_time[i]);

        printf("Burst Time:t");
```

```

scanf("%d", &burst_time[i]);

temp[i] = burst_time[i];
}

printf("\nEnter Time Quantum:t");
scanf("%d", &time_quantum);
printf("\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time\n"); for(total
= 0, i = 0; i != 0;)
{
if(temp[i] <= time_quantum && temp[i] > 0)
{
total = total + temp[i];
temp[i] = 0;
counter = 1;
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - time_quantum;
total = total + time_quantum;
}
if(temp[i] == 0 && counter == 1)
{
x--;
printf("\nProcess[%d]\t\t\t %d\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total -
arrival_time[i] - burst_time[i]);
wait_time = wait_time + total - arrival_time[i] - burst_time[i];
turnaround_time = turnaround_time + total - arrival_time[i]; counter = 0;
}
if(i == limit - 1)
{
i = 0;
}
else if(arrival_time[i + 1] <= total)
{
i++;
}
else
{
}

i = 0;
}
}

average_wait_time = wait_time * 1.0 / limit;
average_turnaround_time = turnaround_time * 1.0 / limit;
printf("\nAverage Waiting Time:t%f", average_wait_time);
printf("\nAvg Turnaround Time:t%f", average_turnaround_time);
return 0;

```

}

### Output:

```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter Total Number of Processes:      4

Enter Details of Process[1]
Arrival Time:  0
Burst Time:    4

Enter Details of Process[2]
Arrival Time:  1
Burst Time:    7

Enter Details of Process[3]
Arrival Time:  2
Burst Time:    5

Enter Details of Process[4]
Arrival Time:  3
Burst Time:    6

Enter Time Quantum:      3

Process ID      Burst Time      Turnaround Time      Waiting Time
Process[1]      4              13                   9
Process[3]      5              16                   11
Process[4]      6              18                   12
Process[2]      7              21                   14

Average Waiting Time:  11.500000
Avg Turnaround Time:  17.000000
```

**Ex. No.: 7**

**Date:**

## **IPC USING SHARED MEMORY**

**Aim:**

To write a C program to do Inter Process Communication (IPC) using shared memory between sender process and receiver process.

**Algorithm:**

### **sender**

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Write a string to the shared memory segment using sprintf
5. Set delay using sleep
6. Detach shared memory segment using shmdt

### **receiver**

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Print the shared memory contents sent by the sender process.
5. Detach shared memory segment using shmdt

**Program Code:**

#### **sender.c**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SharedMemSize 50
void main()
{
    char c;
    int shmid;
    key_t key;

    char *shared_memory;
```



```

key = 5677;
//Create segment with the key specified
if ((shmid = shmget(key, SharedMemSize, IPC_CREAT |
0666)) < 0)
{
    //perror explains error code
    perror("shmget");
    exit(1);
}

//Attach the segment
if((shared_memory= shmat(shmid, NULL, 0)) == (char *) -1) {
    perror("shmat");
    exit(1);
}

sprintf(shared_memory, " Welcome to Shared Memory");
sleep(2);
exit(0);
}

```

#### **receiver.c**

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define SharedMemSize 50

void main()
{
    int shmid;
    key_t key;
    char *shared_memory;
    key = 5677;
    if ((shmid = shmget(key, SharedMemSize, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    //Attach the segment to our data space
    if((shared_memory = shmat(shmid, NULL, 0))==(char *) -1) {
        perror("shmat");
        exit(1);
    }
    //Read the message sender sent to the shared memory
    printf("Message Received: %s \n",shared_memory); exit(0);
}

```

**Output:****Terminal 1**

```
[root@localhost student]# gcc sender.c -o sender  
[root@localhost student]# ./sender
```

**Terminal 2**

```
[root@localhost student]# gcc receiver.c -o receiver  
[root@localhost student]# ./receiver  
Message Received: Welcome to Shared Memory  
[root@localhost student]#
```

**Ex. No.: 8**

**Date:**

### **PRODUCER CONSUMER USING SEMAPHORES**

**Aim:** To write a program to implement solution to producer consumer problem using semaphores.

**Algorithm:**

1. Initialize semaphore empty, full and mutex.
2. Create two threads- producer thread and consumer thread.
3. Wait for target thread termination.
4. Call sem\_wait on empty semaphore followed by mutex semaphore before entry into critical section.
5. Produce/Consume the item in critical section.
6. Call sem\_post on mutex semaphore followed by full semaphore
7. before exiting critical section.
8. Allow the other thread to enter its critical section.
9. Terminate after looping ten times in producer and consumer Threads each.

**Program Code:**

```
#include<stdio.h>
#include<stdlib.h>
```

```
int mutex=1,full=0,empty=3,x=0;
```

```
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                    producer();
                    else
                        printf("Buffer is full!!");

                    break;
            case 2: if((mutex==1)&&(full!=0))
                    consumer();
                    else
                        printf("Buffer is empty!!");
```

```

                                break;
                                case 3:
                                exit(0);
                                break;
                                }
                                }

                                return 0;
                                }

                                int wait(int s)
                                {
                                return (--s);
                                }

                                int signal(int s)
                                {
                                return(++s);
                                }

                                void producer()
                                {
                                mutex=wait(mutex);
                                full=signal(full);
                                empty=wait(empty);
                                x++;
                                printf("\nProducer produces the item %d",x);
                                mutex=signal(mutex);
                                }

                                void consumer()
                                {
                                mutex=wait(mutex);
                                full=wait(full);
                                empty=signal(empty);
                                printf("\nConsumer consumes item %d",x);
                                x--;
                                mutex=signal(mutex);
                                }

```

### Output:

```

1. Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!

```

Enter your choice:1  
Producer produces the item 1  
Enter your choice:1  
Producer produces the item 2  
Enter your choice:1  
Producer produces the item 3  
Enter your choice:1  
Buffer is full!!  
Enter your choice:3

**Ex. No.: 9**

**Date:**

## **DEADLOCK AVOIDANCE**

**Aim:**

To find out a safe sequence using Banker's algorithm for deadlock avoidance.

**Algorithm:**

1. Initialize work=available and finish[i]=false for all values of i
2. Find an i such that both:  
finish[i]=false and Need<sub>i</sub> ≤ work
3. If no such i exists go to step 6
4. Compute work=work+allocation<sub>i</sub>
5. Assign finish[i] to true and go to step 2
6. If finish[i]==true for all i, then print safe sequence
7. Else print there is no safe sequence

**Program Code:**

```
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4

    int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    int avail[3] = { 3, 3, 2 }; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }

    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
```

```

need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {

int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
}

if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}

printf("The SAFE Sequence is \n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);

return (0);

// This code is contributed by Deep Baldha (CandyZack) }

```

### Output:

The SAFE Sequence is  
P1 -> P3 -> P4 -> P0 -> P2

**Ex. No.: 10a)**

**Date:**

### **BEST FIT**

**Aim:**

To implement Best Fit memory allocation technique using Python.

**Algorithm:**

1. Input memory blocks and processes with sizes
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking the further processes.

**Program Code:**

```
#include<stdio.h>
#include<string.h>
void bestFit(int blockSize[], int m, int processSize[], int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size and assign to it
    for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process
        int bestIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }

        // If we could find a block for current process if
        (bestIdx != -1)
        {
```



```

// allocate block j to p[i] process
allocation[i] = bestIdx;

// Reduce available memory in this block.
blockSize[bestIdx] -= processSize[i];
}
}

printf("\nProcess No. \tProcess Size\tBlock no. \n"); for
(int i = 0; i < n; i++)
{
// cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
printf("%d \t\t %d ",i+1,processSize[i]);
if (allocation[i] != -1)
printf("\t\t%d",allocation[i] + 1);
else
printf("\n Not Allocated");
printf("\n");
}
}

// Driver code
int main()
{
int blockSize[] = { 100, 500, 200, 300, 600};
int processSize[] = { 212, 417, 112, 426};
int m = sizeof(blockSize)/sizeof(blockSize[0]); int n =
sizeof(processSize)/sizeof(processSize[0]);
bestFit(blockSize, m, processSize, n);

return 0 ;
}

```

### **Output:**

```

Process No. Process Size Block no.
1 212 4
2 417 2
3 112 3
4 426 5

```

**Ex. No.: 10b)**

**Date:**

### **FIRST FIT**

**Aim:**

To write a C program for implementation memory allocation methods for fixed partition using first fit.

**Algorithm:**

1. Define the max as 25.
- 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. 3: Get the number of blocks,files,size of the blocks using for loop.
- 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
- 5: Check highest

**Program Code:**

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files:-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
```

```

ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}

```

### Output:

```

Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no:      File_size :      Block_no:      Block_size:      Fragment
1             1             1             5             4
2             4             2             8             4
3             7             4             10            3_

```

52

**Ex. No.: 11a)**

**Date:**

### **FIFO PAGE REPLACEMENT**

**Aim:**

To find out the number of page faults that occur using First-in First-out(FIFO) page replacement technique.

**Algorithm:**

1. Declare the size with respect to page length
2. Check the need of replacement from the page to memory
3. Check the need of replacement from old page to new page in memory
4. Form a queue to hold all pages
5. Insert the page require memory into the queue
6. Check for bad replacement and page fault
7. Get the number of processes to be inserted
8. Display the values

**Program Code:**

```
def fifo():
    global a,n,m
    f = -1
    page_faults = 0
    page = []
    for i in range(m):
        page.append(-1)

    for i in range(n):
        flag = 0
        for j in range(m):
            if(page[j] == a[i]):
                flag = 1
                break

        if flag == 0:
            f=(f+1)%m
            page[f] = a[i]
            page_faults+=1
            print "\n%d -> " % (a[i]),
            for j in range(m):
                if page[j] != -1:
                    print page[j],
            else:

        print "-",
        else:
            print "\n%d -> No Page Fault" % (a[i]),
            print "\n Total page faults : %d." % (page_faults)

    a = []
    n = input("\n Enter the size of reference string : ")
    for i in range(n):
        a.append(input(" Enter [%2d] : " % (i+1))) m
```

```
= input("\n Enter page frame size : ") fifo()
```

**Output:**

```
[root@localhost student]# python fifo.py
```

```
Enter the size of reference string : 20
```

```
Enter [ 1] : 7
```

```
Enter [ 2] : 0
```

```
Enter [ 3] : 1
```

```
Enter [ 4] : 2
```

```
Enter [ 5] : 0
```

```
Enter [ 6] : 3
```

```
Enter [ 7] : 0
```

```
Enter [ 8] : 4
```

```
Enter [ 9] : 2
```

```
Enter [10] : 3
```

```
Enter [11] : 0
```

```
Enter [12] : 3
```

```
Enter [13] : 2
```

```
Enter [14] : 1
```

```
Enter [15] : 2
```

```
Enter [16] : 0
```

```
Enter [17] : 1
```

```
Enter [18] : 7
```

```
Enter [19] : 0
```

```
Enter [20] : 1
```

```
Enter page frame size : 3
```

```
7 -> 7 - -
```

```
0 -> 7 0 -
```

```
1 -> 7 0 1
```

```
2 -> 2 0 1
```

```
0 -> No Page Fault
```

```
3 -> 2 3 1
```

```
0 -> 2 3 0
```

```
4 -> 4 3 0
```

```
2 -> 4 2 0
```

```
3 -> 4 2 3
```

```
0 -> 0 2 3
```

```
3 -> No Page Fault
```

```
2 -> No Page Fault
```

```
1 -> 0 1 3
```

```
2 -> 0 1 2
```

```
0 -> No Page Fault
```

```
1 -> No Page Fault
```

```
7 -> 7 1 2
```

```
0 -> 7 0 2
```

```
1 -> 7 0 1
```

```
Total page faults: 15.
```

```
[root@localhost student]#
```

**Ex. No.: 11b)**

**Date:**

### LRU

**Aim:**

To write a c program to implement LRU page replacement algorithm.

**Algorithm:**

- 1: Start the process
- 2: Declare the size
- 3: Get the number of pages to be inserted
- 4: Get the value
- 5: Declare counter and stack
- 6: Select the least recently used page by counter value
- 7: Stack them according the selection.
- 8: Display the values
- 9: Stop the process

**Program Code:**

```
#include<stdio.h>
```

```
int findLRU(int time[], int n){
    int i, minimum = time[0], pos = 0;

    for(i = 1; i < n; ++i){
        if(time[i] < minimum){
            minimum = time[i];
            pos = i;
        }
    }

    return pos;
}

int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults
    = 0;

    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter reference string: ");

    for(i = 0; i < no_of_pages; ++i){
        scanf("%d", &pages[i]);
    }

    for(i = 0; i < no_of_frames; ++i){
        frames[i] = -1;
    }
    for(i = 0; i < no_of_pages; ++i){
```

```

flag1 = flag2 = 0;

for(j = 0; j < no_of_frames; ++j){
    if(frames[j] == pages[i]){
        counter++;
        time[j] = counter;
        flag1 = flag2 = 1;
        break;
    }
}

if(flag1 == 0){
    for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == -1){
            counter++;
            faults++;
            frames[j] = pages[i]; time[j] = counter;
            flag2 = 1;
            break;
        }
    }

    if(flag2 == 0){
        pos = findLRU(time, no_of_frames); counter++;
        faults++;
        frames[pos] = pages[i];
        time[pos] = counter;
    }

    printf("\n");

    for(j = 0; j < no_of_frames; ++j){

        printf("%d\t", frames[j]); }

    }

    printf("\n\nTotal Page Faults = %d", faults);
return 0;
}

```

### Output

```

Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3
5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4

```

**Ex. No.: 11c)**

**Date:**

**Optimal**

**Aim:**

To write a c program to implement Optimal page replacement algorithm.

**ALGORITHM:**

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least frequently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1; int recent[10],optcal[50],count=0;
int optvictm(); void main()
{ clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT
ALGORITHM");
printf("\n..... ");
.....
printf("\nEnter the no.of frames");
scanf("%d",&nof);
printf("Enter the no.of reference string");

scanf("%d",&nor);
printf("Enter the reference string");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n..... ");
printf("\nThe given string");
printf("\n..... \n");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=0;i<nof;i++)
{
```



```

frm[i]=-1;
optcal[i]=0;
}
for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\tref no %d ->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])

{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++; else
victim=optvictim(i);
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
}

printf("\n Number of page faults: %d",pf); getch();
}
int optvictim(int index)
{
int i,j,temp,notfound; for(i=0;i<nof;i++)
{
notfound=1;

for(j=index;j<nor;j++)
if(frm[i]==ref[j])
{
notfound=0;
optcal[i]=j;
break;
}
if(notfound==1) return i;
}
temp=optcal[0];
for(i=1;i<nof;i++)
if(temp<optcal[i])

```

```

temp=optcal[i];
for(i=0;i<nof;i++)
if(frm[temp]==frm[i]) return i;
return 0;
}

```

## OUTPUT:

### OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3

Enter no.of reference string..6

Enter reference string..6 5 4 2 3 1

### OPTIMAL PAGE REPLACEMENT ALGORITHM

The given reference string:

..... 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 2 3 4

Reference NO 1-> 2 3 1

No.of page faults...6

**Ex. No.: 12**

**Date:**

### **File Organization Technique- Single and Two level directory**

#### **AIM:**

To implement File Organization Structures in C are

- a. Single Level Directory
- b. Two-Level Directory
- c. Hierarchical Directory Structure
- d. Directed Acyclic Graph Structure

#### **a. Single Level**

#### **Directory**

#### **ALGORITHM**

1. Start
2. Declare the number, names and size of the directories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories.
5. Stop.

#### **PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm,count,i,j,mid,cir_x;
char fname[10][20];
initgraph(&gd,&gm,"c:\\tc\\bgi");
cleardevice();
setbkcolor(Green);
puts("Enter the number of files");
scanf("%d",&count);
for(i=0;i<count;i++)
{
```

```

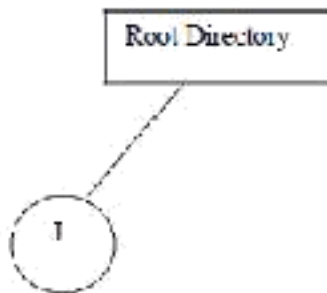
cleardevice();
setbkcolor(GREEN);
printf("Enter the file %d name",i+1);
scanf("%s",fname[i]);

setfillstyle(1,MAGENTA);
mid=640/count; cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextjustify(1,1);
outtextxy(320,125,"Root Directory");
setcolor(BLUE);
for(j=0;j<=i;j++,cir_x+=mid)
{
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
}
}
}

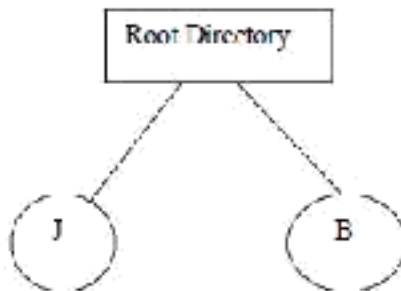
```

#### OUTPUT:

Enter the Number of files  
2  
Enter the file1 J



Enter the file2 B



## **b. Two-level directory Structure**

### **ALGORITHM:**

1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop.

### **PROGRAM:**

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level; struct tree_element *link[5];
}; typedef struct tree_element node;
void main() {
int gd=DETECT,gm; node *root;
root = NULL; clrscr();
create(&root,0,"null",0,630,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\bgi");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node*)malloc(sizeof(node));
printf("enter name of dir/file(under %s):",dname);fflush(stdin);
gets((*root)->name);
if(lev==0||lev==1)
(*root)->ftype=1;
```

else

```
(*root)->ftype=2;
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
if(lev==0||lev==1)
{
if((*root)->level==0)
printf("How many users");
else
printf("How many files");
printf("(for%s):",(*root)->name);
scanf("%d",&(*root)->nc);
}
else(*root)->nc=0;
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root!=NULL)
```

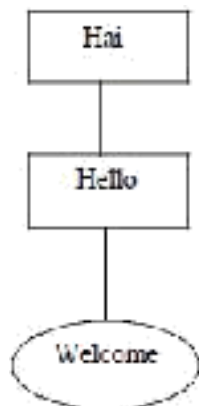
```

{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else
fillellipse(root->x,root->y,20,20); outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}

```

### OUTPUT:

Enter the name of dir/file(under null): Hai  
How many users(for Hai):1  
Enter name of dir/file(under Hai):Hello  
How many files(for Hello):1  
Enter name of dir/file(under Hello):welcome



## Viva Questions

### 1. Installation and Configuration of Linux

1. **What are the basic steps to install Linux?**
    - Boot from the installation media, partition the disk, choose a file system, install the base system, configure system settings, and complete the installation.
  2. **What is GRUB?**
    - GRUB is the bootloader used in Linux systems to manage and boot operating systems.
  3. **Name a few popular Linux distributions.**
    - Ubuntu, Fedora, CentOS, Debian, Arch Linux.
  4. **What is a kernel in Linux?**
    - The kernel is the core part of the Linux operating system that manages hardware and system processes.
  5. **What is the purpose of a swap partition in Linux?**
    - A swap partition is used as virtual memory to extend RAM.
  6. **How do you configure a network during Linux installation?**
    - Using tools like nmtui, ip, or configuring network settings manually.
  7. **What is LVM, and why is it used?**
    - Logical Volume Manager allows flexible disk management by creating, resizing, and managing disk partitions.
  8. **What file system is commonly used in Linux?**
    - Ext4, XFS, Btrfs, etc.
  9. **What is the difference between a live and an installed Linux system?**
    - A live system runs directly from the media without installation, while an installed system runs from the hard drive.
  10. **What is the role of /etc/fstab?**
    - It is used to define how disk partitions and file systems are mounted at boot.
- 

### 2. Basic Linux Commands

1. **How do you list files in a directory?**
  - Use the ls command.
2. **What does the pwd command do?**
  - Displays the present working directory.
3. **How do you create a new directory?**
  - Use the mkdir command.
4. **What is the command to view the contents of a file?**
  - Use cat, less, or more.
5. **How do you copy a file in Linux?**
  - Use the cp command.
6. **How do you delete a directory?**
  - Use rmdir for empty directories or rm -r for non-empty ones.



7. **What is the purpose of the chmod command?**
  - To change file permissions.
8. **How can you display the first 10 lines of a file?**
  - Use the head command.
9. **What does grep do?**
  - Searches for patterns in files.
10. **How do you check disk usage?**
  - Use the df and du commands.

### 3. Study of Unix Editors: sed, vi, emacs

1. **What is sed used for?**
  - It is a stream editor for parsing and transforming text.
2. **What are the modes in vi editor?**
  - Command mode, insert mode, and visual mode.
3. **How do you save and exit in vi?**
  - Press ESC and type :wq.
4. **What is emacs primarily used for?**
  - It is a powerful text editor used for programming and other tasks.
5. **How do you replace text in sed?**
  - Use the s command, e.g., sed 's/old/new/g'.
6. **What is the difference between vi and vim?**
  - vim is an improved version of vi with additional features.
7. **How do you copy lines in vi?**
  - Use yy to copy and p to paste.
8. **What does Ctrl+x in emacs do?**
  - It initiates a command.
9. **How do you search in vi?**
  - Use / followed by the search term.
10. **What is the use of q in sed?**
  - It exits the script after processing a pattern.

### 4. Shell Scripting

#### a) Arithmetic Operations using expr

1. **What is expr in shell scripting?**
  - It evaluates expressions and outputs the result.
2. **How do you add two numbers using expr?**
  - `expr num1 + num2`
3. **What symbol is used for multiplication in expr?**
  - The `\*` symbol.
4. **What is the significance of backticks ( ` ) in shell scripting?**
  - They are used to execute commands and capture their output.
5. **Can expr handle floating-point numbers?**
  - No, expr only works with integers.

6. **How do you subtract numbers using expr?**
  - `expr num1 - num2`
7. **What is the difference between `$((...))` and `expr`?**
  - `$((...))` is faster and supports complex arithmetic.
8. **How can you store the result of `expr` in a variable?**
  - `result=$(expr num1 + num2)`
9. **What happens if you divide by zero using `expr`?**
  - It throws an error.
10. **How do you use parentheses in `expr`?**
  - Use escaped parentheses: `\(` and `\)`.

#### **b) Check Leap Year using if-else**

1. **What is the condition for a year to be a leap year?**
  - It should be divisible by 4 but not by 100 unless also divisible by 400.
2. **How do you use if statements in shell scripting?**
  - `if [ condition ]; then ... fi`
3. **What does `[ ]` in shell scripting mean?**
  - It's used for test expressions.
4. **What is the significance of `-eq` in shell scripting?**
  - It checks if two numbers are equal.
5. **How do you compare strings in shell scripting?**
  - Use `=` or `!=`.
6. **Can a leap year script handle negative years?**
  - Yes, it can if not explicitly restricted.
7. **How do you capture user input in shell?**
  - Use the `read` command.
8. **What is the role of `elif` in shell scripting?**
  - It's used for multiple conditions.
9. **Can you use `case` instead of `if` for this problem?**
  - No, `case` is not suitable for numerical conditions.
10. **What is the exit status of an if statement?**
  - 0 for true and 1 for false.

#### **c) Reverse a Number using while Loop**

1. **What is the syntax of a while loop in shell scripting?**
  - `while [ condition ]; do ... done`
2. **How do you extract the last digit of a number in shell?**
  - Use modulus operator `%`.
3. **How do you divide a number in shell scripting?**
  - Use `expr num / divisor`.
4. **What is the role of `rev` command in Linux?**
  - It reverses strings or numbers.
5. **What happens if you input a negative number in this script?**
  - You need to handle the negative sign explicitly.

6. **What is the difference between while and until loops?**
  - while executes as long as the condition is true, until executes until the condition is true.
7. **Can you reverse numbers with leading zeros?**
  - No, leading zeros are removed.
8. **What is \$# in shell scripting?**
  - The number of arguments passed to the script.
9. **What does \$((...)) do in shell scripting?**
  - It performs arithmetic operations.
10. **Can you use a for loop instead of while here?**
  - No, because the iteration count is not fixed.

#### **d) Fibonacci Series using for Loop**

1. **What is a Fibonacci series?**
  - A series where each number is the sum of the two preceding ones.
2. **What is the syntax of a for loop in shell scripting?**
  - for var in list; do ... done
3. **How do you initialize variables in shell scripting?**
  - Using var=value.
4. **What is the role of echo in the script?**
  - To print output.
5. **How do you calculate the next Fibonacci number?**
  - Add the last two numbers.
6. **What is the use of seq command?**
  - Generates a sequence of numbers.
7. **Can you generate Fibonacci using recursion in shell?**
  - Yes, but it's inefficient in shell scripting.
8. **How do you pass input to a shell script?**
  - Using command-line arguments or read.
9. **What happens if you input a non-integer?**
  - The script will throw an error unless handled.
10. **What does break do in a loop?**
  - It exits the loop immediately.

## **6. System Calls**

### **System Calls: fork(), exec(), getpid(), opendir(), readdir()**

1. **What is a system call?**
  - An interface for user programs to interact with the operating system.
2. **What does fork() do?**
  - Creates a new process (child process).
3. **What is the return value of fork()?**
  - 0 for the child process, PID of the child for the parent.
4. **What does exec() do?**

- Replaces the current process with a new program.
- 5. **What does getpid() return?**
  - The process ID of the current process.
- 6. **What is opendir() used for?**
  - Opens a directory stream.
- 7. **How do you read entries from a directory in C?**
  - Use readdir().
- 8. **What happens if fork() fails?**
  - Returns -1.
- 9. **Can exec() return to the calling process?**
  - No, unless it fails.
- 10. **How do you close a directory stream in C?**
  - Use closedir().

## **7. Scheduling Algorithms**

### **a) FCFS**

1. **What is FCFS?**
  - First-Come, First-Served scheduling.
2. **Is FCFS preemptive or non-preemptive?**
  - Non-preemptive.
3. **What is the main disadvantage of FCFS?**
  - High average waiting time due to convoy effect.

### **b) SJF**

1. **What is SJF?**
  - Shortest Job First scheduling.
2. **Is SJF optimal?**
  - Yes, for minimizing waiting time.
3. **What is the major problem with SJF?**
  - Starvation for longer jobs.

### **c) Priority Scheduling**

1. **What is Priority Scheduling?**
  - Processes are executed based on priority.
2. **What is starvation in scheduling?**
  - Low-priority processes might never execute.

## **d) Round Robin**

- 1. What is Round Robin?**
  - Each process gets an equal time quantum for execution.
- 2. Is Round Robin preemptive?**
  - Yes.

## **8. Inter-Process Communication using Shared Memory**

### **1. What is Inter-Process Communication (IPC)?**

- IPC is a mechanism that allows processes to communicate and share data with each other. It is essential for coordinating actions between multiple processes.

### **2. What is shared memory in IPC?**

- Shared memory is a block of memory that can be accessed by multiple processes. It is the fastest IPC mechanism as it avoids the overhead of kernel-based message passing.

### **3. What system calls are used for shared memory in Linux?**

- The key system calls are:
  - `shmget()` to allocate shared memory.
  - `shmat()` to attach the shared memory segment to a process's address space.
  - `shmdt()` to detach the shared memory segment.
  - `shmctl()` to control shared memory operations (e.g., delete the segment).

### **4. What is the role of `shmget()` in shared memory?**

- `shmget()` is used to create or get access to a shared memory segment. It takes parameters such as a unique key, size of the memory, and permissions.

### **5. What does `IPC_CREAT` flag in `shmget()` do?**

- The `IPC_CREAT` flag is used to create a new shared memory segment if it does not already exist.

### **6. What is the significance of `shmat()` in shared memory?**

- `shmat()` attaches the shared memory segment to the process's address space, allowing the process to read from or write to the shared memory.

### **7. How do you detach a shared memory segment?**

- Use the `shmdt()` system call, which removes the shared memory segment from the process's address space.

## 8. What is the role of `shmctl()` in shared memory?

- `shmctl()` is used to perform control operations on shared memory, such as deleting the shared memory segment or retrieving its information.

## 9. What are the advantages of shared memory for IPC?

- Shared memory is fast because data does not need to be copied between processes. It is efficient for large data transfers and provides direct memory access.

## 10. What are the limitations of shared memory?

- Shared memory lacks built-in synchronization, so mechanisms like semaphores or mutexes must be used to avoid race conditions. Additionally, managing shared memory requires careful handling to avoid memory leaks.

## 9. Producer Consumer using Semaphores.

### 1. What is the Producer-Consumer problem?

- The Producer-Consumer problem is a classic synchronization problem where the producer generates data and puts it into a buffer, and the consumer retrieves data from the buffer. The challenge is to ensure that the producer does not overwrite a full buffer and the consumer does not read from an empty buffer.

### 2. What is a semaphore?

- A semaphore is a synchronization tool used to manage access to shared resources. It can be used to signal and control processes in concurrent programming.

### 3. What are the two types of semaphores?

- **Binary Semaphore:** Also called a mutex, it can have only two values: 0 and 1.
- **Counting Semaphore:** Can have a range of values and is used for resource management.

### 4. What are the key operations on a semaphore?

- **Wait (P operation):** Decreases the semaphore value. If the value is less than 0, the process is blocked.
- **Signal (V operation):** Increases the semaphore value and wakes up a blocked process if any.

### 5. What semaphores are used in the Producer-Consumer problem?

- **Mutex:** Ensures mutual exclusion when accessing the buffer.
- **Full:** Counts the number of items in the buffer.
- **Empty:** Counts the number of empty slots in the buffer.

### 6. How does the Producer work in the Producer-Consumer problem?

- The producer waits on the empty semaphore, locks the buffer using mutex, adds an item to the buffer, unlocks the buffer using mutex, and signals the full semaphore.

### 7. How does the Consumer work in the Producer-Consumer problem?

- The consumer waits on the full semaphore, locks the buffer using mutex, removes an item from the buffer, unlocks the buffer using mutex, and signals the empty semaphore.

8. **What is the role of the buffer in the Producer-Consumer problem?**
  - The buffer is a shared resource where the producer stores data and the consumer retrieves data. It is typically implemented as a bounded (fixed-size) queue.
9. **What happens if the producer tries to produce when the buffer is full?**
  - The producer will be blocked because the empty semaphore value will be 0, indicating that no empty slots are available.
10. **What happens if the consumer tries to consume when the buffer is empty?**
  - The consumer will be blocked because the full semaphore value will be 0, indicating that there are no items to consume.

## 10. Banker's Deadlock Avoidance Algorithm

1. **What is the Banker's Algorithm?**
  - It is a deadlock avoidance algorithm that ensures the system stays in a safe state by allocating resources only if it does not lead to a deadlock.
2. **What is a safe state?**
  - A system state is safe if it can allocate resources to all processes in some order without causing a deadlock.
3. **What are the key data structures used in the Banker's Algorithm?**
  - **Available:** Resources currently available.
  - **Maximum:** Maximum resources a process may need.
  - **Allocation:** Resources currently allocated to processes.
  - **Need:** Remaining resources a process needs.
4. **How do you calculate the Need matrix?**
  - $Need[i][j] = Maximum[i][j] - Allocation[i][j]$ .
5. **What is a request in the context of the Banker's Algorithm?**
  - A request is a demand by a process for additional resources during its execution.
6. **What happens if a resource request cannot be safely granted?**
  - The process is made to wait until the resources are available in a safe manner.
7. **What are the assumptions of the Banker's Algorithm?**
  - The number of resources is fixed, and all processes declare their maximum needs in advance.
8. **What is the difference between deadlock prevention and avoidance?**
  - **Prevention:** Restricts resource allocation to avoid deadlocks entirely.
  - **Avoidance:** Dynamically decides if granting a request will keep the system in a safe state.
9. **Why is the Banker's Algorithm not commonly used in real systems?**
  - It requires knowing maximum resource needs in advance, which may not always be feasible.
10. **What is the time complexity of the Banker's Algorithm?**
  - $O(m \times n^2)$ , where  $m$  is the number of resource types and  $n$  is the number of processes.

## 11. Memory Allocation Techniques

### a) Best Fit

1. **What is the Best Fit allocation strategy?**
  - Allocates the smallest available memory block that fits the request.
2. **What is the advantage of Best Fit?**
  - Minimizes wasted space within allocated blocks.
3. **What is the main disadvantage of Best Fit?**
  - Can lead to external fragmentation.
4. **What is external fragmentation?**
  - Unused memory spaces outside the allocated blocks.
5. **How does Best Fit compare with First Fit?**
  - Best Fit takes longer to find a block but may use memory more efficiently.
6. **Is Best Fit a dynamic or static allocation strategy?**
  - It is a dynamic allocation strategy.
7. **How is the memory block selected in Best Fit?**
  - By searching all available blocks and choosing the smallest that fits.
8. **Can Best Fit cause delays in allocation?**
  - Yes, as it requires searching all available blocks.
9. **How can fragmentation be reduced in Best Fit?**
  - By periodically compaction (defragmentation).
10. **What is the trade-off in Best Fit?**
  - Improved memory utilization at the cost of increased allocation time.

### b) First Fit

1. **What is the First Fit allocation strategy?**
  - Allocates the first available memory block that fits the request.
2. **What is the advantage of First Fit?**
  - Faster allocation compared to Best Fit.
3. **What is the disadvantage of First Fit?**
  - Can lead to more fragmentation compared to Best Fit.
4. **Does First Fit require searching the entire memory list?**
  - No, it stops once a suitable block is found.
5. **How does First Fit handle large allocations?**
  - It may leave behind small unusable gaps, leading to fragmentation.
6. **What is internal fragmentation?**
  - Wasted memory space inside an allocated block.
7. **Is First Fit suitable for real-time systems?**
  - Yes, due to its faster allocation.
8. **Does First Fit perform better with frequent allocations and deallocations?**
  - It depends, but fragmentation may worsen with time.
9. **What is a linked list representation of memory?**
  - A structure where each block points to the next, used in First Fit.
10. **How can First Fit be optimized?**



- By maintaining separate free lists for different block sizes.

## 12. Page Replacement Algorithms

### a) FIFO (First In, First Out)

1. **What is FIFO Page Replacement?**
  - Replaces the oldest page in memory when a page fault occurs.
2. **What is the advantage of FIFO?**
  - Simple to implement.
3. **What is the main disadvantage of FIFO?**
  - Can cause more page faults due to the lack of consideration for page usage frequency.
4. **What is Belady's Anomaly?**
  - In FIFO, increasing the number of frames can sometimes increase the number of page faults.
5. **How is the "oldest page" identified in FIFO?**
  - By maintaining a queue of pages.
6. **Is FIFO optimal for page replacement?**
  - No, it does not account for future usage.
7. **What is a page fault?**
  - It occurs when the required page is not in memory.
8. **What data structure is commonly used for FIFO?**
  - A queue.
9. **Does FIFO work well for all workloads?**
  - No, it may perform poorly for certain access patterns.
10. **How can FIFO be improved?**
  - By combining it with frequency-based strategies.

### b) LRU (Least Recently Used)

1. **What is LRU Page Replacement?**
  - Replaces the page that has not been used for the longest time.
2. **What is the advantage of LRU?**
  - Reduces page faults compared to FIFO by considering past usage.
3. **What is the main disadvantage of LRU?**
  - Requires more computational overhead to track usage history.
4. **How is the "least recently used" page tracked?**
  - Using stacks or timestamps.
5. **Does LRU suffer from Belady's Anomaly?**
  - No, it does not.
6. **What is the time complexity of LRU?**
  - $O(n)$  or  $O(\log n)$  with optimized data structures.
7. **What are the common implementations of LRU?**
  - Using a stack or a doubly linked list.
8. **Can LRU be used for hardware implementations?**

- Yes, but it is complex to implement directly.
- 9. **Why is LRU considered better than FIFO?**
  - It accounts for recent page usage.
- 10. **What is a practical approximation of LRU?**
  - Clock (second-chance) algorithm.

### c) Optimal Page Replacement

1. **What is the Optimal Page Replacement algorithm?**
  - Replaces the page that will not be used for the longest time in the future.
2. **Why is it called "optimal"?**
  - It guarantees the minimum number of page faults.
3. **What is the limitation of the Optimal algorithm?**
  - It requires future knowledge of page references, which is impractical.
4. **What is the advantage of the Optimal algorithm?**
  - It sets a benchmark for other algorithms.
5. **What is the use of the Optimal algorithm in practice?**
  - It is used for evaluation purposes, not in real systems.
6. **How does the Optimal algorithm handle ties?**
  - Any page among the ties can be replaced.
7. **Does the Optimal algorithm work for dynamic workloads?**
  - No, because future access patterns are unpredictable.
8. **What is a real-world approximation of Optimal?**
  - LRU is considered an approximation.
9. **What is the time complexity of the Optimal algorithm?**
  - $O(n \times m)O(n \times m)O(n \times m)$ , where  $n$  is the number of pages and  $m$  is the number of frames.
10. **How is the Optimal algorithm used for comparison?**
  - It provides a theoretical lower bound for page faults.

### File Organization Techniques

#### a) Single-Level Directory

1. **What is a single-level directory?**
  - A flat structure where all files are stored in one directory.
2. **What is the main advantage of a single-level directory?**
  - Simplicity in implementation and management.
3. **What is the main disadvantage of a single-level directory?**
  - Difficult to manage when there are many files or multiple users.
4. **How are files uniquely identified in a single-level directory?**
  - By their unique names.
5. **Is there any hierarchy in a single-level directory?**
  - No, it is a flat structure.
6. **What is a limitation of single-level directories in multi-user systems?**
  - File name conflicts among users.

7. **Can access permissions be implemented in a single-level directory?**
  - Yes, but it adds complexity.
8. **What is fragmentation in the context of single-level directories?**
  - Fragmentation does not occur in the directory structure, but it may in the storage system.
9. **Is a single-level directory scalable?**
  - No, it becomes inefficient as the number of files grows.
10. **How are files located in a single-level directory?**
  - Using a linear search.

## **b) Two-Level Directory**

1. **What is a two-level directory?**
  - A directory structure where each user has their own directory.
2. **What is the advantage of a two-level directory?**
  - Prevents file name conflicts among users.
3. **What is the main disadvantage of a two-level directory?**
  - It is still not scalable for systems with a large number of users or files.
4. **How are files accessed in a two-level directory?**
  - By specifying the user directory and the file name.
5. **Does a two-level directory allow file sharing?**
  - No, file sharing is not directly supported.
6. **What is the difference between absolute and relative paths in a two-level directory?**
  - Absolute paths specify the full path, including the user directory, while relative paths are based on the current working directory.
7. **Can the same file name exist in different user directories?**
  - Yes, each user's directory is independent.
8. **What is the role of a user directory in a two-level directory?**
  - It acts as a container for all files belonging to a specific user.
9. **What is the significance of access control in a two-level directory?**
  - It restricts users from accessing each other's files.
10. **How does a two-level directory improve over a single-level directory?**
  - It provides better organization and avoids file name conflicts.