GALGOTIAS UNIVERSITY
Plot No.2, Sector -17 A, Yamuna Expressway,
Greater Noida, Gautam Buddha Nagar, U.P., India

SCHOOL OF COMPUTING SCIENCE & ENGINEERING

"LAB PRACTICAL FILE"

Course Name: OPERATING SYSTEM

Course Code: E2UC401B

School: SCSE **Year: 2nd** **Semester: 4th**
Session: 2023-2024

Program: B. Tech

Submitted By:	Submitted To:
NEERAJ SINGH **21SCSE1011675**	**Mr. R. Sundar Kumar**

INDEX

S. No.	Exp. Name	Date	Sign
1	Simulate the following CPU scheduling algorithms. a) FCFS b) SJF c) Round Robin d) Priority.	30/01/2023	
2	Write a C program to simulate producer-consumer problem using Semaphores	06/02/2023	
3	Write a C program to simulate the concept of Dining-philosophers problem.	13/02/2023	
4	Simulate MVT and MFT.	20/02/2023	
5	Write a C program to simulate the following contiguous memory allocation Techniques a) Worst fit b) Best fit c) First fit.	06/03/2023	
6	Simulate all page replacement algorithms a)FIFO b) LRU c) OPTIMAL	13/03/2023	
7	Simulate all File Organization Techniques a) Single level directory b) Two level directory	27/03/2023	
8	Simulate all file allocation strategies a) Sequential b) Indexed c) Linked.	03/04/2023	
9	Write a script to translate the string from capital letters to small and small letters to capital using awk command.	10/04/2023	
10	Write a script to do the sorting of given numbers (use command line argument).	17/04/2023	
11	Write a program for process creation using C. (Use of gcc compiler).	24/04/2023	
12	Write a script to reverse a number and string given by user	01/05/2023	
13	Write a script to find the smallest of three numbers as well as largest among three	08/05/2023	
14	Write script that prints names of all sub directories present in the current directory.	22/05/2023	
15	Allocate/free memory to processes in whole pages, find max allocatable pages, incorporate address translation into the program.	29/05/2023	

Exp-1 : Simulate the following CPU scheduling algorithms. a) FCFS b) SJF c) Round Robin d) Priority.

a)FCFS

```
#include<stdio.h>
int main(){
  int n, bt[30], wait_t[30], turn_ar_t[30], av_wt_t = 0, avturn_ar_t = 0, i, j;
  printf("Please enter the total number of processes(maximum 30):"); // the maximum process that be used to
calculate is specified.
  scanf("%d", & n);
  printf("\nEnter The Process Burst Timen");
  for (i = 0; i < n; i++) // burst time for every process will be taken as input
  {
    printf("P[%d]:", i + 1);
    scanf("%d", & bt[i]);
  }
  wait_t[0] = 0;

  for (i = 1; i < n; i++) {
    wait_t[i] = 0;
    for (j = 0; j < i; j++)
      wait_t[i] += bt[j];
  }
  printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
  for (i = 0; i < n; i++) {
    turn_ar_t[i] = bt[i] + wait_t[i];
    av_wt_t += wait_t[i];
    avturn_ar_t += turn_ar_t[i];
    printf("\nP[%d]\t\t%d\t\t\t%d\t\t\t\t%d", i + 1, bt[i], wait_t[i], turn_ar_t[i]);
  }
  av_wt_t /= i;
  avturn_ar_t /= i; // average calculation is done here
  printf("\nAverage Waiting Time:%d", av_wt_t);
  printf("\nAverage Turnaround Time:%d", avturn_ar_t);
  return 0;
}
```

```
Please enter the total number of processes(maximum 30): 2
Enter The Process Burst TimenP[1]: 10
P[2]: 5
Process Burst Time Waiting Time Turnaround Time
P[1] 10 0 10
P[2] 5 10 15
Average Waiting Time: 5
Average Turnaround Time: 12
```

b)SJF

```
#include<stdio.h>
int main() {
  int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
  int sumt = 0, sumw = 0;
  printf("enter the no of processes : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("the arrival time for process P%d : ", i + 1);
    scanf("%d", & at[i]);
    printf("the burst time for process P%d : ", i + 1);
    scanf("%d", & burst_time[i]);
    sum_burst_time += burst_time[i];
  }
  burst_time[9] = 9999;
  for (time = 0; time < sum_burst_time;) {
    smallest = 9;
    for (i = 0; i < n; i++) {
      if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
        smallest = i;
    }
    printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest], time
- at[smallest]);
    sumt += time + burst_time[smallest] - at[smallest];
    sumw += time - at[smallest];
    time += burst_time[smallest];
    burst_time[smallest] = 0;
  }
  printf("\n\n average waiting time = %f", sumw * 1.0 / n);
  printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
  return 0;
}
```

```
enter the no of processes: 2
the arrival time
for process P1: 10
the burst time
for process P1: 5
the arrival time
for process P2: 6
the burst time
for process P2: 3
P[10] | -22765 | -32764
the average waiting time = -16382.000000
the average turnaround time = -11382.500000
```

c)Round Robin

```
#include<stdio.h>
int main() {
  int cnt, j, n, t, remain, flag = 0, tq;
  int wt = 0, tat = 0, at[10], bt[10], rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d", & n);
  remain = n;
  for (cnt = 0; cnt < n; cnt++) {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :", cnt + 1);
    scanf("%d", & at[cnt]);
    scanf("%d", & bt[cnt]);
    rt[cnt] = bt[cnt];}
  printf("Enter Time Quantum:\t");
  scanf("%d", & tq);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for (t = 0, cnt = 0; remain != 0;) {
    if (rt[cnt] <= tq && rt[cnt] > 0) {
      t += rt[cnt];
      rt[cnt] = 0;
      flag = 1;
    } else if (rt[cnt] > 0) {
      rt[cnt] -= tq;
      t += tq;}
    if (rt[cnt] == 0 && flag == 1) {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n", cnt + 1, t - at[cnt], t - at[cnt] - bt[cnt]);
      wt += t - at[cnt] - bt[cnt];
      tat += t - at[cnt];
      flag = 0;}
    if (cnt == n - 1)
      cnt = 0;
    else if (at[cnt + 1] <= t)
      cnt++;
    else
      cnt = 0;}
  printf("\nAverage Waiting Time= %f\n", wt * 1.0 / n);
  printf("Avg Turnaround Time = %f", tat * 1.0 / n);
  return 0;}
```

```
Enter Total Process: 4
Enter Arrival Time and Burst Time
for Process Process Number 1: 0 5
Enter Arrival Time and Burst Time
for Process Process Number 2: 1 4
Enter Arrival Time and Burst Time
for Process Process Number 3: 2 2
Enter Arrival Time and Burst Time
for Process Process Number 4: 4 1
Enter Time Quantum: 2
Process | Turnaround Time | Waiting Time
P[3] | 4 | 2
P[4] | 3 | 2
P[2] | 10 | 6
P[1] | 12 | 7
Average Waiting Time = 4.250000
Avg Turnaround Time = 7.250000
```

d)Priority

```
#include <stdio.h>
void swap(int * a, int * b) {
  int temp = * a;
  * a = * b;
  * b = temp;}
int main() {
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d", & n);
    int b[n], p[n], index[n];
    for (int i = 0; i < n; i++) {
      printf("Enter Burst Time and Priority Value for Process %d: ", i + 1);
      scanf("%d %d", & b[i], & p[i]);
      index[i] = i + 1;}
    for (int i = 0; i < n; i++) {
      int a = p[i], m = i;
      for (int j = i; j < n; j++) {
        if (p[j] > a) {
          a = p[j];
          m = j;}}
      swap( & p[i], & p[m]);
      swap( & b[i], & b[m]);
      swap( & index[i], & index[m]);}
    int t = 0;
    printf("Order of process Execution is\n");
    for (int i = 0; i < n; i++) {
      printf("P%d is executed from %d to %d\n", index[i], t, t + b[i]);
      t += b[i];}
    printf("Process Id Burst Time
        int wait_time = 0;
        for (int i = 0; i < n; i++) {
          printf("P%d \t %d \t\t %d
            wait_time += b[i];
          }
          Wait Time TurnAround Time\ n "); %
            d\ n ",index[i],b[i],wait_time,wait_time + b[i]);
          return 0;
}
```

```
Enter Total Number of Process: 2
Enter Burst Time and Priority
P[1]
Burst Time: 12
Priority: 3
P[2]
Burst Time: 45
Priority: 1
Process Burst Time Waiting Time Turnaround Time
P[2] 45 0 45
P[1] 12 45 57
Average Waiting Time = 22
Average Turnaround Time = 51
```

2) Write a C program to simulate producer-consumer problem using Semaphores

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, data = 0;
void producer() {
  --mutex;
  ++full;
  --empty;
  data++;
  printf("\nProducer produces item number: %d\n", data);
  ++mutex;
}
void consumer() {
  --mutex;
  --full;
  ++empty;
  printf("\nConsumer consumes item number: %d.\n", data);
  data--;
  ++mutex;
}
int main() {
  int n, i;
  printf("\n1. Enter 1 for Producer"
    "\n2. Enter 2 for Consumer"
    "\n3. Enter 3 to Exit");
  for (i = 1; i > 0; i++) {
    printf("\nEnter your choice: ");
    scanf("%d", & n);
    switch (n) {
    case 1:
      if ((mutex == 1) && (empty != 0)) {
        producer();
      }
      else {
        printf("The Buffer is full. New data cannot be produced!");
      }
      break;
    case 2:
      if ((mutex == 1) && (full != 0)) {
        consumer();
      }
      else {
        printf("The Buffer is empty! New data cannot be consumed!");
      }
      break;
    case 3:
      exit(0);
      break;
    }
  }
}
```

```
1. Enter 1
for Producer
2. Enter 2
for Consumer
3. Enter 3 to Exit
Enter your choice: 1
Producer produces item number: 1
Enter your choice: 2
Consumer consumes item number: 1.
Enter your choice: 3
```

3) Write a C program to simulate the concept of Dining-philosophers problem.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define NUM_PHILOSOPHERS 5
#define NUM_CHOPSTICKS 5
void dine(int n);
pthread_t philosopher[NUM_PHILOSOPHERS];
pthread_mutex_t chopstick[NUM_CHOPSTICKS];
int main() {
  int i, status_message;
  void * msg;
  for (i = 1; i <= NUM_CHOPSTICKS; i++) {
    status_message = pthread_mutex_init( & chopstick[i], NULL);
    if (status_message == -1) {
      printf("\n Mutex initialization failed");
      exit(1);
    }
  }
  for (i = 1; i <= NUM_PHILOSOPHERS; i++) {
    status_message = pthread_create( & philosopher[i], NULL, (void * ) dine,
(int * ) i);
    if (status_message != 0) {
      printf("\n Thread creation error \n");
      exit(1);
    }
  }
  for (i = 1; i <= NUM_PHILOSOPHERS; i++) {
    status_message = pthread_join(philosopher[i], & msg);
    if (status_message != 0) {
      printf("\n Thread join failed \n");
      exit(1);
    }
  }
  for (i = 1; i <= NUM_CHOPSTICKS; i++) {
    status_message = pthread_mutex_destroy( & chopstick[i]);
    if (status_message != 0) {
      printf("\n Mutex Destroyed \n");
      exit(1);
    }
  }
  return 0;
}
void dine(int n) {
  printf("\nPhilosopher % d is thinking ", n);
  pthread_mutex_lock( & chopstick[n]);
  pthread_mutex_lock( & chopstick[(n + 1) % NUM_CHOPSTICKS]);
  printf("\nPhilosopher % d is eating ", n);
  sleep(3);
  pthread_mutex_unlock( & chopstick[n]);
  pthread_mutex_unlock( & chopstick[(n + 1) % NUM_CHOPSTICKS]);
  printf("\nPhilosopher % d Finished eating ", n);
}
```

```
Philosopher  1 is thinking
Philosopher  1 is eating
Philosopher  3 is thinking
Philosopher  2 is thinking
Philosopher  5 is thinking
Philosopher  3 is eating
Philosopher  4 is thinking
Philosopher  1 Finished eating
Philosopher  2 is eating
Philosopher  4 is eating
Philosopher  5 is eating
Philosopher  3 Finished eating
Philosopher  2 Finished eating
Philosopher  4 Finished eating
Philosopher  5 Finished eating
```

4) Simulate MVT and MFT.

mvt

```
#include<stdio.h>
#include<conio.h>
main(){
int ms,mp[10],i, temp,n=0;
char ch = 'y';
clrscr();
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n++){
printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
scanf("%d",&mp[i]);
if(mp[i]<=temp){
printf("\nMemory is allocated for Process %d ",i+1);
temp = temp - mp[i];
}
else
{
printf("\nMemory is Full");
break;
}
printf("\nDo you want to continue(y/n) -- ");
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);
printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++)
printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp);
 getch();}
```

```
Enter the total memory available (in Bytes) -- 1000
Enter memory required for process 1 (in Bytes) -- 400
Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) -- 550
Memory is Full
Total Memory Available -- 1000
PROCESS MEMORY-ALLOCATED
1                400
2                275
Total Memory Allocated is 675
Total External Fragmentation is 325
```

mft

```
#include<stdio.h>
#include<conio.h>
main()
{
int ms, bs, nob, ef,n, mp[10],tif=0;
int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t%d",bs-mp[i]);
tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
getch();
}
```

```
Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes)-- 300
Enter the number of processes – 5
Enter memory required for process 1 (in Bytes) -- 275
Enter memory required for process 2 (in Bytes) -- 400
No. of Blocks available in memory -- 3
PROCESS MEMORY-REQUIRED ALLOCATED INTERNAL-FRAGMENTATION
1                 275               YES               25
2                 400               NO                -----
Memory is Full, Remaining Processes cannot be accommodated
Total Internal Fragmentation is 42
Total External Fragmentation is 100
```

5) Write a C program to simulate the following contiguous memory allocation Techniques a) Worst fit b) Best fit c) First fit.

a)**Worst Fit**

```
#include <stdio.h>
void implimentWorstFit(int blockSize[], int blocks, int processSize[], int
processes) {
  int allocation[processes];
  int occupied[blocks];
  for (int i = 0; i < processes; i++) {
    allocation[i] = -1;
  }
  for (int i = 0; i < blocks; i++) {
    occupied[i] = 0;
  }
  for (int i = 0; i < processes; i++) {
    int indexPlaced = -1;
    for (int j = 0; j < blocks; j++) {
      if (blockSize[j] >= processSize[i] && !occupied[j]) {
        if (indexPlaced == -1)
          indexPlaced = j;
        else if (blockSize[indexPlaced] < blockSize[j])
          indexPlaced = j;
      }
    }
    if (indexPlaced != -1) {
      allocation[i] = indexPlaced;
      occupied[indexPlaced] = 1;
      blockSize[indexPlaced] -= processSize[i];
    }
  }
  printf("\nProcess No.\tProcess Size\tBlock no.\n");
  for (int i = 0; i < processes; i++) {
    printf("%d \t\t\t %d \t\t\t", i + 1, processSize[i]);
    if (allocation[i] != -1)
      printf("%d\n", allocation[i] + 1);
    else
      printf("Not Allocated\n");
  }
}
int main() {
int blockSize[] = {100,50,30,120,35};
int processSize[] = {40,10,30,60};
  int blocks = sizeof(blockSize) / sizeof(blockSize[0]);
  int processes = sizeof(processSize) / sizeof(processSize[0]);
  implimentWorstFit(blockSize, blocks, processSize, processes);
  return 0;
}
```

```
Process No.Process Size Block no.
1 40 4
2 10 1
3 30 2
4 60 Not Allocated
```

b)Best Fit

```
#include <stdio.h>
void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses) {
  int allocation[proccesses];
  int occupied[blocks];
  for (int i = 0; i < proccesses; i++) {
    allocation[i] = -1;
  }
  for (int i = 0; i < blocks; i++) {
    occupied[i] = 0;
  }
  for (int i = 0; i < proccesses; i++) {
    int indexPlaced = -1;
    for (int j = 0; j < blocks; j++) {
      if (blockSize[j] >= processSize[i] && !occupied[j]) {
        if (indexPlaced == -1)
          indexPlaced = j;
        else if (blockSize[j] < blockSize[indexPlaced])
          indexPlaced = j;
      }
    }
    if (indexPlaced != -1) {
      allocation[i] = indexPlaced;
      occupied[indexPlaced] = 1;
    }
  }
  printf("\nProcess No.\tProcess Size\tBlock no.\n");
  for (int i = 0; i < proccesses; i++) {
    printf("%d \t\t\t %d \t\t\t", i + 1, processSize[i]);
    if (allocation[i] != -1)
      printf("%d\n", allocation[i] + 1);
    else
      printf("Not Allocated\n");
  }
}
int main() {
  int blockSize[] = {100, 50,30,120,35};
  int processSize[] = {40,10,30, 60};
  int blocks = sizeof(blockSize) / sizeof(blockSize[0]);
  int proccesses = sizeof(processSize) / sizeof(processSize[0]);
  implimentBestFit(blockSize, blocks, processSize, proccesses);
  return 0;
}
```

```
Process No.Process Size Block no.
1 10 2
2 30 1
3 60 4
4 30 4
```

c)First Fit

```
#include <stdio.h>
void implimentFirstFit(int blockSize[], int blocks, int processSize[], int processes) {
  int allocate[processes];
  int occupied[blocks];
  for (int i = 0; i < processes; i++) {
    allocate[i] = -1;
  }
  for (int i = 0; i < blocks; i++) {
    occupied[i] = 0;
  }
  for (int i = 0; i < processes; i++) {
    for (int j = 0; j < blocks; j++) {
      if (!occupied[j] && blockSize[j] >= processSize[i]) {
        allocate[i] = j;
        occupied[j] = 1;

        break;
      }
    }
  }
  printf("\nProcess No.\tProcess Size\tBlock no.\n");
  for (int i = 0; i < processes; i++) {
    printf("%d \t\t\t %d \t\t\t", i + 1, processSize[i]);
    if (allocate[i] != -1)
      printf("%d\n", allocate[i] + 1);
    else
      printf("Not Allocated\n");
  }
}
void main() {
  int blockSize[] = {30,5,10};
  int processSize[] = {10,6,9};
  int m = sizeof(blockSize) / sizeof(blockSize[0]);
  int n = sizeof(processSize) / sizeof(processSize[0]);
  implimentFirstFit(blockSize, m, processSize, n);
}
```

```
Process No.Process Size Block no.
1 10 1
2 6 3
3 9 Not Allocate
```

6) Simulate all page replacement algorithms a)FIFO b) LRU c) OPTIMAL

a)FIFO

```
#include<stdio.h>
int main() {
int incomingStream[] = {4,1,2,4,5};
int pageFaults = 0;
int frames = 3;
int m, n, s, pages;
pages = sizeof(incomingStream) / sizeof(incomingStream[0]);
printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
int temp[frames];
for (m = 0; m < frames; m++) {
   temp[m] = -1;
}
for (m = 0; m < pages; m++) {
s = 0;
for (n = 0; n < frames; n++) {
  if (incomingStream[m] == temp[n]) {
    s++;
    pageFaults--;
  }
}
pageFaults++;
if ((pageFaults <= frames) && (s == 0)) {
  temp[m] = incomingStream[m];
} else if (s == 0) {
  temp[(pageFaults - 1) % frames] = incomingStream[m];
}
printf("\n");
printf("%d\t\t\t", incomingStream[m]);
for (n = 0; n < frames; n++) {
  if (temp[n] != -1)
    printf(" %d\t\t\t", temp[n]);
  else
    printf(" - \t\t\t");
}
}
  printf("\nTotal Page Faults:\t%d\n", pageFaults);
  return 0;
}
```

```
Incoming Frame 1 Frame 2 Frame 3
4       4       -       -
1       4       1       -
2       4       1       2
4       4       1       2
5       5       1       2
Total Page Faults: 4
```

b)LRU

```
#include<stdio.h>
#include<limits.h>
int checkHit(int incomingPage, int queue[], int occupied) {
  for (int i = 0; i < occupied; i++) {
    if (incomingPage == queue[i])
      return 1;
  }
  return 0;}
void printFrame(int queue[], int occupied) {
  for (int i = 0; i < occupied; i++)
    printf("%d\t\t\t", queue[i]);
}
int main() {
  int incomingStream[] = {1,2,3,2,1,5,2,1,6,2,5,6,3,1,3};
  int n = sizeof(incomingStream) / sizeof(incomingStream[0]);
  int frames = 3;
  int queue[n];
  int distance[n];
  int occupied = 0;
  int pagefault = 0;
  printf("Page\t Frame1 \t Frame2 \t Frame3\n");
  for (int i = 0; i < n; i++) {
    printf("%d:  \t\t", incomingStream[i]);
    if (checkHit(incomingStream[i], queue, occupied)) {
      printFrame(queue, occupied);
    }
    else if (occupied < frames) {
      queue[occupied] = incomingStream[i];
      pagefault++;
      occupied++;
      printFrame(queue, occupied);
    } else {
      int max = INT_MIN;
      int index;
      for (int j = 0; j < frames; j++) {
        distance[j] = 0;
        for (int k = i - 1; k >= 0; k--) {
          ++distance[j];
          if (queue[j] == incomingStream[k])
            break;
        }
        if (distance[j] > max) {
          max = distance[j];
          index = j;
        }
      }
      queue[index] = incomingStream[i];
      printFrame(queue, occupied);
      pagefault++;
    }
    printf("\n");
  }
  printf("Page Fault: %d", pagefault);
  return 0;}
```

```
Page    Frame1          Frame2          Frame3
1:      1
2:      1               2
3:      1               2               3

2:      1               2               3

1:      1               2               3

5:      1               2               5

2:      1               2               5

1:      1               2               5

6:      1               2               6

2:      1               2               6

5:      5               2               6

6:      5               2               6

3:      5               3               6

1:      1               3               6

3:      1               3               6

Page Fault: 8
```

c)Optimal

```
#include<stdio.h>
int main() {
  int n, pg[30], fr[10];
  int count[10], i, j, k, fault, f, flag, temp, current, c, dist, max, m, cnt, p, x;
  fault = 0;
  dist = 0;
  k = 0;
  printf("Enter the total no pages:\t");
  scanf("%d", & n);
  printf("Enter the sequence:");
  for (i = 0; i < n; i++)
    scanf("%d", & pg[i]);
  printf("\nEnter frame size:");
  scanf("%d", & f);
  for (i = 0; i < f; i++) {
    count[i] = 0;
    fr[i] = -1; }
  for (i = 0; i < n; i++) {
    flag = 0;
    temp = pg[i];
    for (j = 0; j < f; j++) {
      if (temp == fr[j]) {
        flag = 1;
        break;
      }}
    if ((flag == 0) && (k < f)) {
      fault++;
      fr[k] = temp;
      k++;
    } else if ((flag == 0) && (k == f)) {
      fault++;
      for (cnt = 0; cnt < f; cnt++) {
        current = fr[cnt];
        for (c = i; c < n; c++) {
          if (current != pg[c])
            count[cnt]++;
          else
            break;
        }}
      max = 0;
      for (m = 0; m < f; m++) {
        if (count[m] > max) {
          max = count[m];
          p = m;
        } }
      fr[p] = temp; }
    printf("\npage  %d  frame\t", pg[i]);
    for (x = 0; x < f; x++) {
      printf("%d\t", fr[x]);
    }}
  printf("\nTotal number of faults=%d", fault);
  return 0;
}
```

```
Enter the total no pages:    2
Enter the sequence:1 2
Enter frame size:2
page  1  frame     1          -1

page  2  frame     1          2

Total number of faults=2
```

7) Simulate all File Organization Techniques a) Single level directory b) Two level directory

A)Single level

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
  int nf = 0, i = 0, j = 0, ch;
  char mdname[10], fname[10][10], name[10];
  clrscr();
  printf("Enter the directory name:");
  scanf("%s", mdname);
  printf("Enter the number of files:");
  scanf("%d", & nf);
  do {
    printf("Enter file name to be created:");
    scanf("%s", name);
    for (i = 0; i < nf; i++) {
      if (!strcmp(name, fname[i]))
        break;
    }
    if (i == nf) {
      strcpy(fname[j++], name);
      nf++;
    } else
      printf("There is already %s\n", name);
    printf("Do you want to enter another file(yes - 1 or no - 0):");
    scanf("%d", & ch);
  }
  while (ch == 1);
  printf("Directory name is:%s\n", mdname);
  printf("Files names are:");
  for (i = 0; i < j; i++)
    printf("\n%s", fname[i]);
  getch();
}
```

```
Enter the directory name:sss
Enter the number of files:3
Enter file name to be created:aaa
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:bbb
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:ccc
Do you want to enter another file(yes - 1 or no - 0):0
Directory name is:sss
Files names are:
aaa
bbb
ccc
```

b)Two level

```
#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);}}}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");    }
getch();     }
```

```
enter number of directories:1
enter directory 1 names:qwerty
enter size of directories:2
enter subdirectory name and size:hello
1
enter file name:qw
enter subdirectory name and size:hi
2
enter file name:rt
enter file name:er
dirname         size    subdirname      size    files
qwerty          2       hello           1       qw

                        hi              2       rt
        er
```

8) Simulate all file allocation strategies a) Sequential b) Indexed c) Linked.

a)sequential

```
#include<stdio.h>
#include<conio.h>
main() {
  int n, i, j, b[20], sb[20], t[20], x, c[20][20];
  clrscr();
  printf("Enter no.of files:");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("Enter no. of blocks occupied by file%d", i + 1);
    scanf("%d", & b[i]);
    printf("Enter the starting block of file%d", i + 1);
    scanf("%d", & sb[i]);
    t[i] = sb[i];
    for (j = 0; j < b[i]; j++)
      c[i][j] = sb[i]++;
  }
  printf("Filename\tStart block\tlength\n");
  for (i = 0; i < n; i++)
    printf("%d\t %d \t%d\n", i + 1, t[i], b[i]);
  printf("Enter file name:");
  scanf("%d", & x);
  printf("File name is:%d", x);
  printf("length is:%d", b[x - 1]);
  printf("blocks occupied:");
  for (i = 0; i < b[x - 1]; i++)
    printf("%4d", c[x - 1][i]);
  getch();
}
```

```
Enter no.of files:1
Enter no. of blocks occupied by file11
Enter the starting block of file10
Filename        Start block      length
1          0        1
Enter file name:neeraj
File name is:-1length is:1blocks occupied:
```

b) Indexed

```
#include<stdio.h>
main() {
  int n, m[20], i, j, sb[20], s[20], b[20][20], x;

  printf("Enter no. of files:");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("Enter starting block and size of file%d:", i + 1);
    scanf("%d%d", & sb[i], & s[i]);
    printf("Enter blocks occupied by file%d:", i + 1);
    scanf("%d", & m[i]);
    printf("enter blocks of file%d:", i + 1);
    for (j = 0; j < m[i]; j++)
      scanf("%d", & b[i][j]);
  }
  printf("\nFile\t index\tlength\n");
  for (i = 0; i < n; i++) {
    printf("%d\t%d\t%d\n", i + 1, sb[i], m[i]);
  }
  printf("\nEnter file name:");
  scanf("%d", & x);
  printf("file name is:%d\n", x);
  i = x - 1;
  printf("Index is:%d", sb[i]);
  printf("Block occupied are:");
  for (j = 0; j < m[i]; j++)
    printf("%3d", b[i][j]);
}
```

```
Enter no.of files: 2
Enter starting block and size of file1: 2 5
Enter blocks occupied by file1: 10
enter blocks of file1: 3
2 5 4 6 7 2 6 4 7
Enter starting block and size of file2: 3 4
Enter blocks occupied by file2: 5
enter blocks of file2: 2 3 4 5 6
File index length
1 2 10
2 3 5
Enter file name: venkat
file name is: 12803
Index is: 0B lock occupied are:
```

c) Linked

```
#include<stdio.h>
#include<conio.h>
struct file {
  char fname[10];
  int start, size, block[10];
}
f[10];
main() {
  int i, j, n;
  clrscr();
  printf("Enter no. of files:");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("Enter file name:");
    scanf("%s", & f[i].fname);
    printf("Enter starting block:");
    scanf("%d", & f[i].start);
    f[i].block[0] = f[i].start;
    printf("Enter no.of blocks:");
    scanf("%d", & f[i].size);
    printf("Enter block numbers:");
    for (j = 1; j <= f[i].size; j++) {
      scanf("%d", & f[i].block[j]);
    }
  }
  printf("File\tstart\tsize\tblock\n");
  for (i = 0; i < n; i++) {
    printf("%s\t%d\t%d\t", f[i].fname, f[i].start, f[i].size);
    for (j = 1; j <= f[i].size - 1; j++)
      printf("%d--->", f[i].block[j]);
    printf("%d", f[i].block[j]);
    printf("\n");
  }
  getch();
}
```

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6 5 4 3 2
File start size block
venkat 20 6 4--->12--->15--->45--->32--->25
rajesh 12 5 6--->5--->4--->3--->2
```

9) Write a script to translate the string from capital letters to small and small letters to capital using awk command.

```
#!/bin/awk -f

{
  for (i = 1; i <= length($0); i++) {
    char = substr($0, i, 1)
    if (char >= "a" && char <= "z") {
      printf("%s", toupper(char))
    } else if (char >= "A" && char <= "Z") {
      printf("%s", tolower(char))
    } else {
      printf("%s", char)
    }
  }
  printf("\n")
}
```

```
$ echo "Hello World" | awk - f translate.awk
hELLO wORLD
```

10) Write a script to do the sorting of given numbers (use command line argument).

```
#!/bin/bash
# Check
if at least one argument is provided
if [$ # - lt 1];
then
echo "Usage: $0 <number1> <number2> ..."
exit 1
fi

# Store command line arguments in an array
numbers = ("$@")

# Sort the numbers using the 'sort'
command
sorted_numbers = $(printf "%s\n"
  "${numbers[@]}" | sort - n)

# Print the sorted numbers
echo "Sorted numbers:"
echo "$sorted_numbers"
```

```
$. / sort_numbers.sh 10 5 8 3 1
Sorted numbers:
  1
3
5
8
10
```

11) Write a program for process creation using C. (Use of gcc compiler).

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
  pid_t pid; // Variable to store the process ID

  // Fork a child process
  pid = fork();

  if (pid < 0) {
    fprintf(stderr, "Fork failed\n");
    return 1;
  } else if (pid == 0) {
    // Child process
    printf("Child process\n");
    printf("Child process ID: %d\n", getpid());
    printf("Parent process ID: %d\n", getppid());
  } else {
    // Parent process
    printf("Parent process\n");
    printf("Parent process ID: %d\n", getpid());
    printf("Child process ID: %d\n", pid);
  }

  return 0;
}
```

```
Parent process
Parent process ID: 2190
Child process ID: 2191
Child process
Child process ID: 2191
Parent process ID: 1
```

12) Write a script to reverse a number and string given by user

```
#!/bin/bash

# Reverse a number
reverse_number() {
  echo "$1" | rev
}

# Reverse a string
reverse_string() {
  echo "$1" | rev
}

# Prompt
for a number and reverse it
read - p "Enter a number: "
input_number
reverse_number "$input_number"

# Prompt
for a string and reverse it
read - p "Enter a string: "
input_string
reverse_string "$input_string"
```

```
$ bash reverse_simple.sh
Enter a number: 12345
54321
Enter a string: Hello World
dlroW olleH
```

13) Write a script to find the smallest of three numbers as well as largest among three

```
#!/bin/bash
# Find the smallest number among three
find_smallest() {
  smallest = $1
  if [$2 - lt $smallest];
  then
  smallest = $2
  fi
  if [$3 - lt $smallest];
  then
  smallest = $3
  fi
  echo "Smallest number: $smallest"
}
# Find the largest number among three
find_largest() {
  largest = $1
  if [$2 - gt $largest];
  then
  largest = $2
  fi
  if [$3 - gt $largest];
  then
  largest = $3
  fi
  echo "Largest number: $largest"
}
# Prompt
for three numbers
read - p "Enter first number: "
num1
read - p "Enter second number: "
num2
read - p "Enter third number: "
num3
# Find the smallest and largest numbers
find_smallest "$num1"
"$num2"
"$num3"
find_largest "$num1"
"$num2"
"$num3"
```

```
$ bash find_numbers.sh
Enter first number: 10
Enter second number: 5
Enter third number: 8
Smallest number: 5
Largest number: 10
```

14) Write script that prints names of all sub directories present in the current directory.

```
#!/bin/bash

# Print names of subdirectories in current directory
print_subdirectories() {
    for directory in */; do
        echo "$directory"
    done
}

# Call the function to print subdirectory names
print_subdirectories
```

```
$ bash print_subdirectories.sh
subdirectory1/
subdirectory2/
subdirectory3/
```

15) Allocate/free memory to processes in whole pages, find max allocatable pages, incorporate address translation into the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#define PAGE_SIZE 4096
#define MAX_PAGES 10
int main() {
   unsigned long page_count = 0;
   void* address;
   while (page_count < MAX_PAGES) {
      // Allocate a page
      address = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
      if (address == MAP_FAILED) {
         perror("Failed to allocate memory");
         break;
      }
      unsigned long virtual_address = (unsigned long) address;
      unsigned long page_number = virtual_address / PAGE_SIZE;
      unsigned long offset = virtual_address % PAGE_SIZE;
      unsigned long physical_address = (page_number * PAGE_SIZE) + offset;
      printf("Virtual Address: 0x%lx\t Page Number: %lu\t Offset: %lu\t Physical Address: 0x%lx\n",
virtual_address, page_number, offset, physical_address);
      page_count++;
   }
   for (unsigned long i = 0; i < page_count; i++) {
      munmap(address, PAGE_SIZE);
   }
   return 0;
}
```

```
Virtual Address: 0x7f7218eea000     Page Number: 34210942698  Offset: 0  Physical Address: 0x7f7218eea000
Virtual Address: 0x7f7218eb0000     Page Number: 34210942640  Offset: 0  Physical Address: 0x7f7218eb0000
Virtual Address: 0x7f7218eaf000     Page Number: 34210942639  Offset: 0  Physical Address: 0x7f7218eaf000
Virtual Address: 0x7f7218eae000     Page Number: 34210942638  Offset: 0  Physical Address: 0x7f7218eae000
Virtual Address: 0x7f7218ead000     Page Number: 34210942637  Offset: 0  Physical Address: 0x7f7218ead000
Virtual Address: 0x7f7218eac000     Page Number: 34210942636  Offset: 0  Physical Address: 0x7f7218eac000
Virtual Address: 0x7f7218eab000     Page Number: 34210942635  Offset: 0  Physical Address: 0x7f7218eab000
Virtual Address: 0x7f7218c7f000     Page Number: 34210942079  Offset: 0  Physical Address: 0x7f7218c7f000
Virtual Address: 0x7f7218c7e000     Page Number: 34210942078  Offset: 0  Physical Address: 0x7f7218c7e000
Virtual Address: 0x7f7218c7d000     Page Number: 34210942077  Offset: 0  Physical Address: 0x7f7218c7d000
```