PAGE NO:
DATE:

CPU scheduling using FCFS and SJF method

```
#include <stdio.h>
int i,n,j,nos, temp, choice, total=0
int Burst_time[20], Arr_time [20], wait_time[20],
Turnar_time [20], process [20];
float avg_time = around_time [0] avg_wait_time=0

void FCFS()
{
  int tot_wai_time =0, tot_turnar_time
      =0;
  int cur_time=0;

  for (i=0; i<n-1; i++){
      for (j=i+1; j<n; j++){
        if (j=i+1; j<n; j++){
          temp = Arr time [i];
          Arr time = Arr_time [j];
          Arr_time [j]= temp;

          temp = Burst_time [i]
          Burst time [j]= Burst_time [i]
          Burst_time [j]= temp;
```

```
        temp = process[i];
        process[i] = process[j];
        process[j] = temp;
    }
  }
}

wait_time[0] = 0;
cur_time = Arr_time[0] + Burst_time[0];

for (i=1; i<n; i++){
    if (cur_time < Arr_time[i]){
        cur_time = Arr_time[i];
    }
    wait_time[i] = cur_time - Arr_time[i];
    cur_time += Burst_time[i];
    tot_wait_time += wait_time[i];
}

printf("\nProcess\t\tArrival Time\t\tWait Time
        \t\tTurn Around time");

for (i=0; i<n; i++){
    Turn_aro_time[i] = Burst_time[i] +
                        wait_time[i];
    tot_turn_time += Turn_aro_time[i]
    printf("\nP[%d]\t\t%d\t\t%d\t\t%d",
        process[i], Arrival_time[i],
        Burst_time[i], wait_time[i],
        Turn_aro_time[i]);
```

```
avg_wait_time = (float)total_wait_time/n
avg_Turn_aro_time = (float)total_turnaro_
                    time/n
printf("\n Average Wait Time : %.2f",
        avg_wait_time);
printf("\n Average Turnaround time : %.2f\n",
        avg_Turn_around_time);

}

void SJF()
{
    int total_wait_time = 0, total_turnaro_time=0;
    int completed = 0, cur_time=0, min_index;
    int is_completed[20]={0};

    while (completed != n){
        int min_bur_time = 999;
        min_index = -1;

        for (i=0; i<n; i++){
            if (Arr_time[i] <= cur_time &&
                is_completed[i] == 0){
                if (Burst_time
                min_index = i;
            }
        }
    }
}
```

```
if (min_index != 1){
    wait_time[min_index] = cur_time -
                Arr_time[min_index]

    cur_time += Burst_time [min_index]
    Tur_aro_time [min_index] = cur_time -
                Arr_time[min_index]

    total_wait_time += waiting_time [min_index]
    total_aro_time += Turn_around_time [min_index]

    is_completed [min_index] = 1;
    completed++;
}
else {
    cur_time ++;
}}

printf("\nProcess\t\t Arrival time \t
        Burst time \t waiting_time[i],
        Tur_around_time [i]);

avg_waiting_time = (float) total_wait_time/n
avg_Turn_aro_time = (float) total_turnaro_
                time /n;
printf("\n\n Average time = %.2f",
        avg_wait_time);
printf("\n Avg Turnaround time = %.2f\n",
        avg_turn_around_time)
}
```

```
int main(){
    printf("Enter the total Number of
            Processes :");
    scanf("%d", &n);
    printf("\n Enter Arrival time &
            Burst time ");
    for (i=0; i<n; i++){
        printf("P[%d] Arr_time : ", i+1);
        scanf("%d", &Arr_time[i]);
        printf("P[%d], Burst time : ", i+1);
        scanf("%d", &Burst_time[i]);
        process[i] = i+1;
    }
    while (1) {
        printf("Main Menu \n");
        printf(" 1. FCFS Scheduling \n 2. SJF
                scheduling \n");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: FCFS();
                break;
            case 2: SJF();
                break;
            default: printf("Invalid input");
        }
    }
    return 0
}
```

Output

Enter no of process : 3
Enter Burst time :
P[1] = 2
P[2] = 1
P[3] = 3

Main Menu

1) FCFS
2) SJS
~~3) F~~

Enter choice : 1

Process	Burst time	Waiting time	Turnaround time
P[1]	2	0	2
P[2]	1	2	3
P[3]	3	3	6

Avg waiting time = 1.67
Avg Turnaround time = 3.67

Enter choice 2

Process	Burst Time	Waiting	Turnaround time
P[2]	1	0	1
P[1]	2	1	3
P[3]	3	3	6

Avg waiting time = 1.33
Avg Turnaround time = 3.33
Enter choice : 3

15/5/26

Process scheduling

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

struct process {
    int pro_id;
    int bur_id;
    int priority;
    int arr_time;
    int wai_time;
    int tur_time;
};

void find_wait_time (struct process proc[],
                     int n, int wt[])
{
    wt[0]=0;
    for (int i=1; i<n; i++){
        wt[i] = proc[i].bur_time +
                wt[i-1];
    }
}

void find_turnaround_time (struct process pro[],
                           int n, int wt[],
                           int tat[])
{
    for (int i=0; i<n; i++){
        tat[i] = pro[i].bur_time + wt[i]
    }
}

void find_average_time (struct process [],
                        int n) {
    int wt[10], tat[10], total_wt=0
    total_tat=0;
    find_turnaround_time(pro, n, wt, tat);
    printf("\n Process ID \t Burst time \t
           Priority \t Arrival Time \t waiting
           Time \t Turnaround Time ");
    for (int i=0; i<n; i++) {
        total_wt = total_wt + wt[i]
        total_wt = total_tat + tat[i]
        printf("\n %d \t \t \t %d \t %d
               proc[i].pro_id,
               proc[i].bur_time, proc[i].p
               iority, proc[i].arr_time,
               wt[i], tat[i]);
    }
    printf("\n\n Average waiting Time = %f",
           (float) total_wt / n);
    printf("\n Average Turnaround time
           %f \n", wt[i], tat[i]);
```

```
void priority_scheduling (struct process pro[],
                          int n)
{
    struct process temp;
    for (int i=0; i<n; i++){
        int pos = i;
        for (int j = i+1; j<n; j++){
            if (pro[j].priority < pro[pos].
            priority || (pro[i].priority ==
                         pro[pos].priority &&
        pos = j;
    }

    temp = pro[i]
    pro[i] = pro[pos]
    pro[pos] = temp;
}
    Fin average time (pro, n)
```

output

```
Enter No. of process: 5
Enter the process ID = 1
Enter burst time = 5
Enter the priority = 2

Enter PID ID: 2
Enter Burst time 3
```

```
Enter process ID: 4
Enter burst time: 5
Enter priority

Enter process ID: 5
Enter burst time 5
Enter the priority: 5
```

Process ID	Burst time	Pri	WT	Turn Time
1	4	2	0	4
2	3	3	4	7
3	1	4	7	8
4	5	5	8	13
5	2	5	13	15

Avg waiting time: 6.40000
Avg Turnaround time: 2.40000

Round Robin (Non Pre-emptive)

```
#include <stdio.h>
#include <stdbool.h>
int turnaroundtime (int proc[], int n, int bt[],
                    int wt[], int tat[])
{
    for (int i=0; i<n; i++)
    {
        tat[i] = bt[i] + wt[i]
        return 1;
```

```
int waitTime(int prs[], int bt, int wt[],
            int q, quantum) {
    int rem = bt[n];
    for(int i=0; i<n; i++)
    {
      rembt[i] = bt[i];
      int t=0;
      while (1)

        bool done = true;
        for (int i=0; i<n; i++)
        {
          if( rem bt[i] > quantum) {
            done = false;
          if (rem bt[i] > quantum)
          {
            t = t = quantum;
            rem bt = quantum;
          }
        else {
            t = t + rembt[i];
            wt[i] = t bt[i];
            rem bt[i] = 0;
          }
      }
    }
    if (done == true) break;
  }
  return 1;

int findavgTime (int prs[], int bt[], int n,
                int quantum)
{
```

```
    waitime (prs, n, bt, wt, quantum);
    turaroutime (prs, n, bt, wt, tat);
    printf (" Process \t\t\t Burst time \t\t\t Wait time
            \t\t\t TA Time \n");
    for (int i=0; i<n; i++)
    {
      totalwt = taraw wt + wt[i];
      total tat = total tat + tat[i];
      printf(" \n \t %d \t\t\t %d \t\t\t %d \t %d \n"
            i+1, bt[i], wt[i], tat[i]);
    }
    printf (" Avg wait Time = %f", (float) totalwt
            / (float)n);
    return 1
}

int main()
{
    int n;
    printf(" Enter no of process : ");
    Scanf (" %d", &n);
    int prs, int burtime[n];
    printf (" Enter quantum ");
    scanf (" %d", &quantum);
    int i=0;
    for (i=0; i<n; i++) {
        printf (" Enter the pro: ");
        Scanf (" %d", &prs[i]);
        printf ("Enter BT ");
        Scanf (" %d", &burtime[i]);
    }
    find av
```

```
finishing time (pos, n, burst time, quantum);
    return 0;
}
```

Output:

```
Enter the number of pos: 5
Enter quantum time: 3
Enter pos: 1
Enter BT = 4
Enter pos = 2
Enter BT = 3
Enter pos = 3
Enter BT = 1
Enter pos = 4
Enter BT = 5
Enter the pos: 5
Enter the BT = 2
```

pos	BT	WT	TA Time
1	4	9	13
2	3	3	6
3	1	6	7
4	5	10	15
5	2	10	12

Rate monotonic and Earliest Deadlight

```
~~#include~~
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

#define maxpos 10
typedef struct {
        int ID;
        int BT;
        float pri;
} Task;

int numpos;
int exetime[maxpos], nerwo[maxpos],
        remtime[maxpos], deadline[maxpos],
        rem_d[maxpos];

void get_pos_info(int selalgo) {
    printf("Enter total number of process (maximum
        %d)", maxpos);
    scanf("%d", &numpos);
    if (numpos < 1)
    {
        exit(0);
    }
```

```
for (int i=0; i<numpcs; i++){
    printf("\nProcess %d \n", i+1);
    printf(" ===> Execution time ");
    scanf("%d", exetime[i]);
    temtime[i] = exetime[i];
    if (selected_algo = 2) {
        printf(" ==> DL ");
        scanf("%d", &deadline[i]);
    }
    else
    {
    printf(" ==> period: ");
    scanf("%d", &period[i]);
    }
}
}
int max[int a, b, c]
{
    int max
    if (a>=b && a>=c)
        max = a
    else if (b>=a && b>=c)
        max=b
    else if (c>a || c>=b)
        max=c
    return max
}
int get_abs_time (int sel_algo)
{
    if (sel_algo==1)
    {
    }
```

```
    return max (period[0], period[1],
                period[2]);
    }
    else if (sel_algo==2)
    {
        return max (deadline[0], deadline[1],
                    deadline[2]);
    }
}

void print_schedule (int pos, st[], int cycle)
{
    print ("\scheduling \n\n");
    printf(" Time: ");
    for (int i=0; i<max_q_pos; i++)
    {
        printf(" P[%d] : ", i+1);
        for (int j=0; j<cycle; j++)
        {
            if (pos_out[j] == i+1)
                printf("1");
            else
                printf("1");
        }
        printf("\n");
    }
}

void rate_monotonic (int time)
{
    int px_i[10]={0}, min=999,
    period DL=0;
    float util=0;
```

```
for(int i=0; i<num_of_pr; i++)
{
    util += (1.0* exe_time[i]/period[i])
}
int n = num_of_pros;
int m = (float) (n*pow(2, 1/n)-1);
if (util > m)
{
    printf("Given problem is not schedule
            under fro rate mono schedule
            algorithm\n");
}
for(int i=0; i<time; i++)
{
    min = 1000;
    for (int j=0; i<num_of_pr; j++)
    {
        if (rem_time[j] > 0)
        {
            if(min > period[j])
            {
                min = period[j];
                next = pr = j;
            }
        }
    }
if (rem_time[next_pr] > 0)
process_st[i] = next_process[r];
rem_time[next_pr] = i;
}
```

```
if (rem_time[next_pr] > 0)
{
    prstime[i] = next_pr+1;
    rem_time[next_pr] = 1
}
for (int k=0; k<num_of_pr; i++)
{
    if (i+1) % period[k] == 0)
    {
        rem_time[k] = exe_time[k];
        next_pr = k;
    }
}
}
printf("schedule (process_list, time)
}
void cor_dl_first (int time) {
    float uti = 0;
    for (int i=0; i<num_of_pr; i++){
        uti = (1.0*exe_time[i]/
                deadline[i]);
}
int n = num_of_pr;
int pr[num_of_pr];
int max_d, cur_process, min_dl,
process_list[time];
bool is_ready[num_of_pr];

for (int i=0; i<num_of_pr; i++)
    is_ready[i] = true;
    pr[i] = i+1;
}
```

```
switch (option)
{
Case 1: get_px_info (option)
        observation_time = get_obs_time
                           (option)
        rate_monotonic (observation_time)
        break;
Case 2: get_px_info (option)
        obs_time = get_obs_time (option)
        earliest_dl_first (observation_time)
        break;
Case 3: exit(0);
default: printf("Invalid")
}
}while(1)
}
```

Output

1) Rate monotonic 2: Earliest Deadline
Enter choice 2, Enter total time & pxs?

p2 1

Execution time: 3 => deadline 5

p2 2
=> Execution time => 2 => Ded 5

p2 3
=> Execution time => 2 => deadline = 10

Schedule

time	00	01	02	03	04	05	06	07	08	09
p1					#			#	#	
p2	#	#					#	#		
p3			#	#						

Write a program to simulate producer consumer using semaphores

```
#include <stdio.h>
#include <stdlib.h>
int mex=1, full=0, empty=3, x=0;

int main()
{
  int n;
  int prod();
  void cons();
  int wait(int);
  int signal(int);
  printf("1. Producer \n2. Consumer
         3. exit");
  while(1) {
  printf("\nEnter your choice");
  scanf("%d", &n);
  switch(n)
  {
    Case 1: if((mex==1)&&(empty!=0))
              prod();
              else
              printf("Buffer is full!!")
              break;
    Case 2: if(mex==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 prod()
{
  mex = wait(mex)
  full = signal(full);
  empty = wait(empty);
  x++;
  printf("\nProducer produces the item %d",
         x);
  mex = signal(mex);
}
void cons()
{
  mex = wait(mex);
  full = wait(full);
  empty = signal(empty);
  printf("\nConsumer consumes item %d", x);
  x--;
  mex = signal(mex);
}
```

Output

1. Producer
2. Consumer
3. Exit

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 : 2
Consumer consumes item 3

Enter your choice : 3

While 0 (program is made to accept
4 Dinning Philosophers

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

#define N 5
#define THK 2
#define HNGRY 1
#define EAT 0
#define Left(i+1)%N
#define Right(i+1)%N

int state[N];
int phil[N] = {0,1,2,3,4};

Semt mex;
Semt s[N];

void test(int i)
{
    if (state[i]==HNGRY && state[Left i] != EAT && state[Right] != EAT)
    {
        state[i] = EAT;
        sleep(2);
        printf("Philosopher %d takes
                fork %d and %d \n", i+1,
                Left(i+1), i+1);
        printf("Philosopher %d Eating \n",
                i+1);
        Sempost(&s[i]);
    }
```

Output

1. Producer
2. Consumer
3. Exit

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 : 2
Consumer consumes item 1

Enter your choice : 3

Write a C program to implement the concept of Dining - Philosophers

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

#define N 5
#define THINK 2
#define HUNGRY 1
#define EATING 0
#define Left(i+N)%N
#define Right(i+1)%N

int state[N];
int phil[N] = {0,1,2,3,4};

Semt mtex;
Semt s[N];

void test(int i)
{
    if (state[i] == HUNGRY && state[Left(i)] != EATING &&
        state[Right(i)] != EATING)
    {
        state[i] = EATING;
        sleep(2);
        printf("Philosopher %d takes
            fork %d and %d\n", i+1,
            Left(i)+1, i+1);
        printf("Philosopher %d is Eating\n",
            i+1);
        Sempost(&s[i]);
    }
```

```
void take_fork(int i)
{
    sem_wait(&mutex);
    state[i] = HUNGRY;
    printf("Philosopher %d is hungry\n",
            i+1);
    test(i);
    sem_post(&mutex);
    sem_wait(&s[i]);
    sleep(1);
}

void put_fork(int i)
{
    sem_wait(&mutex);
    state[i] = THK;
    printf("Philosopher %d putting fork
            and %d down\n", i+1, left+1,
            i+1);
    printf("Philosopher %d is thinking\n", i+1);
    test(left);
    test(right);
    sem_post(&mutex);
}

void *philosopher(void* num)
{
    while(1)
    {
        int *i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex,0,1);
    for(i=0; i<N; i++)
    {
        pthread_create(&thread_id[i],
                NULL, philosopher,
                &phil[i]);
        printf("Philosopher %d is thinking\n",
                i+1);
    }
    for(i=0; i<N; i++)
    {
        pthread_join(thread_id[i], NULL);
    }
}
```

output

phil 1 is Thinking
phil 2 is Thinking
phil 3 is Thinking
phil 4 is Thinking
phil 5 is Thinking
phil 1 is Hungry
phil 2 is Hungry
phil 3 is Hungry
phil 4 is Hungry
phil 5 is Hungry
phil 1 takes fork 1 and 5
phil 1 is eating
phil 2 takes fork 1 and 2
phil 2 is eating
phil 3 takes fork 2 and 3
phil 3 is eating
phil 4 takes fork 3 and 5
phil 4 is eating
phil 5 takes fork 4 and 5
phil 5 is eating
phil 1 is putting fork 5 and 1 down
phil 1 is thinking
phil 2 is putting 1 and 2 down
phil 2 is thinking
phil 3 is putting 2 and 3 down
phil 3 is thinking
phil 4 is putting 3 and 4 down
phil 4 is thinking
phil 5 is putting 5 and 4 down
phil 5 is thinking

Write a C program to simulate Banker algorithm for the purpose of deadlock avoidance

```
#include <stdio.h>
int main ()
{
  int n,m,i,j,k;
  printf("Enter the Process num: ");
  scanf("%d", &n);
  printf("Enter the resources num: ");
  scanf("%d", &m);
  int alloc[n][m];
  printf("Enter allocation matrix");
  for(i=0; i<n; i++)
  {
    for(j=0; j<m; j++)
    {
      scanf("%d", &alloc[i][j]);
    }
  }
  int max[n][m];
  printf("Enter the max matrix");
  for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      scanf("%d", &max[i][j]);

  int avlb[m];
  printf("Enter the available resources: \n");
  for(i=0; i<m; i++)
    scanf("%d", &available[i]);
```

```
int f[n], ans[n], index;
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<n; k++)
    for (i=0; i<n; i++
        if (f[i]==0)
            int flag=0;
            for (j=0; j<m; j++)
                if (need[i][j] > avlb[j])
                    flag=1;
                    break;
        if (flag==0)
            ans[index++] = i;
            for (y=0; y<m; y++)
                avlb[y]+= alloc[i][y];
            f[i]= 1;

int flag= 1;
for (i=0; i<n; i++)
    if (f[i]==0)
        flag = 0;
        printf("The following system is not safe\n");
        break;
if (flag==1)
    printf("Following is the the safe sequence \n");
    for (i=0; i<n-1; i++)
        printf("P%d->", ans[i]);
    printf("P%d\n", ans[n-1]);
return 0
}
```

Output

```
Enter the process num: 5
Enter the resource num: 3
0 1 0  2 0 0  3 0 2  2 1 1  0 0 2
Enter the MAX matrix
7 5 3  3 2 2  9 0 2  2 2 2  4 3 3
Enter the available resources
3 2 2
Following is the safe sequence:
P1 -> P3 -> P4 -> P0 -> P2
```

Write a C program to simulate deadlock detection

```
#include <stdio.h>
static int max[20];
int i, j, n, p, nr;
int main()
{
    int alloc[7][10], req[10][10], avail[10],
      r[10], w[10];

    printf("No of processes")
    scanf("%d", &np);
    printf("Enter no of resources");
    scanf("%d", &nr);
    for (i=0; i<nr; i++)
    {
        printf("\n Total cnt of resource R %d", i+1);
        scanf("%d", &r[i]);
    }

    printf("Enter the request matrix :");
    for (i=0; i<np; i++)
    for (j=0; j<nr; j++)
    scan("%d", &req[i][j]);

    printf("Enter the allocation matrix :")
    for(i=0; i<np; i++)
    for(j=0; j<nr; j++)
    scanf("%d", &alloc[i][j]);
    for(j=0; j<nr; j++)
    {
        avail[j]= r[j];
        for (i=0; i<np; i++){
            avail[j]= alloc[i][j]
        }
    }

    for( i=0; i<np; i++)
    {
        int count=0;
        for (j=0; j<nr; j++)
        {
            if(alloc[i][j]==0)
                count++;
            else
                break;
        }
        if(count==nr)
        mark[i]=1;

        for( j=0; j<nr; j++
            w[j]= avail[j];
        for (i=0; i<np; i++)
            int canbeprs=0;
            if (mark[i]!=1)
            {
                for (j=0; j<nr; j++)
                {
```

```
        canbepos = 1;
    else
    {
        canbepos = 0;
        break;
    }
}
if (canbepos)
{
    mark[i] = 1;
    for(j=0; j< n; j++)
    w[j] += alloc[i][j];
}
}
}

int deadlock = 0;
for( i=0; i<np; i++)
if (mark[ i ] = 1)
deadlock = 1;
if (deadlock)
printf("\n Deadlock detected);
else
printf(" No deadlock possible");
}
```

Output

```
Enter no of processes 5
Enter no of resources = 3
Total Amount of Resource R1: 7
Total Amount of Resource R2: 5
Total Amount of Resource R3: 3

Enter request matrix
210 202 002 100 010

Enter allocation matrix
213 020 302 111 000

Deadlock detected
```

16/04

Write a C program for contiguous memory

(a) Worst - fit
(b) Best - Fit
(c) First - Fit

```
#include <stdio.h>
#define max 25

void bf(int b[], int nb, int f[], int nf)
void wf(int b[], int nb, int f[], int nf)
void ff(int b[], int nb, int f[], int nf)

int main()
{
  int b[max], f[max], nb, nf
  printf("Memory management schemes\n")
  printf("Enter the no of blocks:")
  scanf("%d", &nb);
  printf("Enter no of files");
  scanf("%d", &nf);
  printf("Enter the size of the blocks:\n");
  for(int i = 1; i<=nb; i++)
  {
    printf("Block %d", i);
    scanf("%d", &b[i]);
  }

  printf("Memory management scheme - First fit")
  ff(b, nb, f, nf);
```

```
  printf("\n\n Memory management system - Worst fit");
  wf(b, nb, f, nf);
  printf("\n\n Memory management System - best fit");
  bf(b, nb, f, nf);

  return 0
}

void ff(int b[], int nb, int f[], int nf)
{
  int bf[max]={0};
  int ff[max]={0};
  int frag[max], i, j;

  for(i=1; i<=nf; i++)
    for(j=1; j<=nb; j++)
      if(bf[j]==1 && b[j] >= f[i])
        {
          ff[i]=j
          bf[j]=1
          frag[i]=b[j]-f[i]
          break;

  printf("File no:\t File size:\t Block no:\t Block size:\t Fragment");
  for(i=1; i<=nf; i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
```

```
void wf(int b[], int nb, int f[], int nf)
{
    int bf[max]={0};
    int ff[max]={0};
    int farr[max], i,j, tp, hgh=0;
    for(i=1; i<=nf; i++)
        for(j=1; j<=nb; j++)
            if(bf[j]!=1)
                temp=b[j]-f[i];
                if(temp>=0 && highest < temp)
                {
                    bf[j]=j;
                    hgh= tp;
                }
    farr[i]=hgh;
    bf[ff[i]]=1
    hgh=0;

void bf(int b[], int nb, int f[], int nf)
{
    int bf[max]={0};
    int ff[max]={0};
    int farr[max], i,j, lp, twl=10000;

    for(i=1; i<=nf; i++)
        for(j=1; j<=nb; j++)
            if(bf[j]!=1)
            {
                ff[i]=j;
                lw l= tp;
            }
    farr[i]=dwl;
    bf[ff[i]]=1;
    lwl = 10000;

    printf("\nFile no:\t File size:\t
            Block no:\t Block size\t
            Fragment:");
    for(i=1; i<=nf; ff[i]=0; i++)
    {
        printf("\n%d\t\t %d\t\t %d\t\t
            %d\t\t %d", i, f[i], ff[i],
            b[ff[i]], farr[i]);

    }
```

output:

~~Enter~~

Enter the no of blocks
5
Enter no of processes 8
Enter block size
100 500 200 300 600
Enter the parcel size
212 417 63 124 23 89 73 13
1. First Fit
2. Best Fit
3. Worst Fit
Enter your choice
2

Parcel No.	Parcel A size	Block size
1	212	2
2	417	5
3	63	1

4	125	2
5	23	1
6	89	2
7	79	2
8	13	1

1. First Fit
2. Best Fit
3. Worst Fit

Enter your choice 2

Proc. No	Process Size	Block
1	212	4
2	417	NA
3	63	4
4	125	5
5	23	4
6	89	3
7	79	3
8	13	3

1. First Fit 2. Best Fit 3. Worst Fit

Enter your choice 3

Proc No	Process Size	Block No
1	212	NA
2	415	NA
3	63	NA
4	125	NA
5	23	5
6	89	NA
7	79	NA
8	13	5

Write a C program for page replacement

(a) FIFO
(b) LRU
(c) Optimal

~~#include <stdio.h>~~

```
#include <stdio.h>
int n, b, i, j, k;
int in[50]
int p[50];
int hit=0;
int ~~pg~~ pg6=0;

void getData()
{
    printf("\nEnter length of page reference
            sequence ");
    scanf("%d", &n);
    printf("\n Enter page ref no : ");
    for(i=0;i<n;i++)
        scanf("%d", &in[i]);
    printf("\nEnter no of frames: ");
    scanf("%d", &b);
}

void initialize()
{
    pg6=0;
    for(i=0;i<b;i++)
```

```
        p[i] = 9999;
}

int isHit(int data)
{
    hit = 0;
    for (j=0; j<6; j++)
        if (p[j] == data)
            hit = 1;
            break;
    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<6; k++)
        if (p[k] == data)
            hitind = k;
            break;
    return hitind;
}

void DispPage()
{
    for(k=0; k<6; k++)
    {
        if(p[k] != 9999)
            printf("%d", p[k])
}
}
```

```
void dispPgfaultCnt()
{
    printf("\nTotal no of page fault: %d",
            pgfaultcnt);
}

void fifo()
{
    getdata();
    initialize();
    for(i=0; i<n; i++)
    {
        printf("For %d: ", in[i]);
        if (isHit(in[i]) == 0)
        {
            for (k=0; k<6-1; k++)
                p[k] = p[k+1];
            p[k] = in[i]
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgfaultCnt();

void optimal()
{
    initialize()
    int near[10];
    for(i=0; i<n; i++)
    {
```

```
printf("\nFor: %d : ",in[i]);

if(isfull(in[i]) == 0)
  for(j=0; j<fs; j++)
    int pg = p[j];
    int found = 0;
    for(k=i; k<n; k++)
        if(pg == in[k])
            near[j] = k;
            found = 1;
            break;
    else
        found = 0;
    if(!found)
        near[j] = 9999;
  }
  int max = 9999;
  int repindex;
  for(j=0; j<fs; j++)
      if(near[j] > max)
          max = near[j];
          repindex = j;
  p[repindex] = in[i];
  pgfault(cnt++);
  dispPages();
}
else
  printf("No page fault");
}
dispPgFault(cnt);
}
```

```
void lru()
{
  initialize();
  int least[fs];
  for(i=0; i<n; i++)
  {
    printf("\nFor %d : ",in[i]);
    if(isfull(in[i]) == 0)
    {
      for(k=i-1; k>=0; k--)
          if(pg == in[k])
              least[j] = k;
              found = 1;
              break;
          else
              found = 0;
      }if(!found)
          least[j] = -9999;
      int min = 9999;
      int repindex;
      for(j=0; j<fs; j++)
          if(least[j] < min)
              min = least[j];
              repindex = j;
      p[repindex] = in[i];
      pgfault(cnt++);

      dispPages();
    }
    else
```

```
        printf("No page fault");
    }
    displayPgfaultcnt();
}

int main()
{
    int choice;
    while(1)
    {
        printf("\nPage Replacement Algo \n
                1. Enter data \n
                2. FIFO \n 3. Optimal \n
                4. LRU \n 5. Exit \nExit
                your choice: );
        scanf("%d", &choice);
        switch(choice)
        {
            Case 1: getData();
                    break;
            Case 2: fifo()
                    break;
            Case 3: optimal()
                    break;
            Case 4: lru()
                    break
            default: return 0
                    break;
        }
    }
}
```

Output

```
Page Replacement Algorithm
1) Enter data
2) FIFO
3) Optimal
4) LRU
5) Exit.
Enter your choice: 2
Enter length of page references sequence: 12
Enter the page sequence. 1234 125 1234 5
Enter no of frames: 3

For 1: 1
For 2: 12
For 3: 123
For 4: 234
For 1: 341
For 2: 412
For 5: 125
For 1: No page fault
For 2: No page fault
~~For 3: 125~~
For 3: 253
For 4: 534
For 5: No page fault
Total no of page fault: 9
```

Page Replacement Algorithms
Enter data
FIFO
Optimal
LRU
Exit
Enter your choice : 3

For 1: 1
For 2: 12
For 3: 123
For 4: 124
For 1: No page fault
For 2: No page fault
For 5: 125
For 1: No page fault
For 2: No page fault
For 3: 325
For 4: 425
For 5: No page fault
Total No. of page fault : 7

Page Replacement Algorithm
Enter data
FIFO
Optimal
LRU
Exit
Enter your choice : 4

For 1: 1
For 2: 12
For 3: 123
For 4: 423
For 1: 413
For 2: 412
For 5: 512
For 1: No page fault
For 2: No page fault
For 3: 312
For 4: 342
For 5: 345
Total No. of page faults : 10

Page replacement Algorithm
Enter Data
FIFO
Optimal
LRU
Exit
Enter your choice : 5

3/7/24

Write a C program to simulate Disk scheduling Algo

FCFS

(a)

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int RQ[100], i, n, Total Head Moment = 0, ini;
    printf ("Enter the no of requests \n");
    scanf ("%d", &n);
    printf ("Enter the requests sequence \n");
    for (i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf ("Enter initial head position ");
    scanf ("%d", &ini);

    for (i=0; i<n; i++)
        THM = THM + abs(RQ[i] - ini);
        ini = RQ[i];
    printf ("Total head movement is %d", THM);
    return
}
```

```
Enter no of requests 8
Enter the request sequence
98 183 37 122 14 124 65 67
Enter initial head position 53
Total head moment is 640
```

(b)

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int RQ[100], i, j, n, THM = 0, ini, size, mv;
    printf ("Enter the no of Requests \n")
    scanf ("%d", &n);
    printf ("Enter the Requests sequence")
    for (i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf ("Enter initial head position \n").
    scanf ("%d", &ini);
    printf ("Enter total disk size \n")
    scanf ("%d", &size);
    printf ("Enter the head movement direction for
            high 1 and for low 0 \n");
    scanf ("%d", &move);

    for (i=0; i<n; i++)
        for (j=0; j<n-i-1; j++)
            if (RQ[j] > RQ[j+1])
                int temp;
```

```
temp = RQ[j];
RQ[j] = RQ[j+1];
RQ[j+1] = temp;

if index:
for(i=0; i<n; i++)
    if (initial<RQ[i])
        index = i;
        break;

if (move==1)
    for (i=index; i<n; i++)
    { THM = THM + abs(RQ[i] - ini);
        ini = RQ[i];

THM = THM + abs(size-RQ[(n-1)] - 1);
ini = size-1;
for (i=index-1; i>=0; i--)
    THM = THM + abs(RQ[i] - initial);
    initial = RQ[i];

else
    for (i=index-1; i>=0; i--)
        THM = THM + abs(RQ[i+1] - 0);
        ini = RQ[i];

THM = THM + abs(RQ[i+1] - 0);
ini = 0;
for (i=index; i<n; i++)
    THM = THM + abs(RQ[i] - ini);
    ini = RQ[i];
printf("THM is %d", THM);
}
return 0;
```

Output

```
Enter no of requests :
Enter the request sequence
98 183 37 122 14 124 65 67
Enter initial head position
53
Enter total disk size: 99
Enter the head movement direction for
1 for low 0
0
Total head movement 5236
```

(c)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100], i, j, n, THM=0, ini, size, move;
    printf("Enter the no of requests");
    scanf("%d", &n);
    printf("Enter the request sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &ini);
    printf("Enter total disk size\n");
    scanf("%d", &move);
```

```
for(i=0; i<n; i++)
    for(j=0; j<n-i-1; j++)
        if(RQ[j]>RQ[j+1])
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;

int index;
for(i=0; i<n; i++)
    if(ini<RQ[i])
        index=i;
        break;

if(move==1)
    for(i=index; i<n; i++)
        THM=THM+abs(RQ[i]-ini);
        ini=RQ[i];
    THM=THM+abs(size-RQ[i-1]-1)
    THM=THM+abs(size-1-0);
    ini=0;
    for(i=0; i<index; i++)
        THM=THM+abs(RQ[i]-ini)
        ini=RQ[i];

else
    for(i=index-1; i>=0; i--)
        THM=THM+abs(RQ[i]-ini);
        ini=RQ[i];
    THM=THM+abs(size-1-0);
    ini=size-1
```

```
for(i=n-1; i>=index; i--)
    THM=THM+abs(RQ[i]-ini);
    ini=RQ[i];

printf("THM is %d", THM);
return 0;

}
```

OUTPUT:

```
Enter the number of requests
8
Enter the request sequence
98 183 37 122 14 124 65 67
Enter the initial head position
53
Enter total disk size
199
Enter the head movement dir. for high
1 for low 0
0
Total head movement is 331
```

10/9/24