NAME: ADITYA SINGH

STD: 4th Sem DIV: A ROLL NO.: 1BM22CS022

SUBJECT: OS (Lab)

YOUVA

INDEX

SR. NO.	DATE	TITLE	PAGE NO.	TEACHER'S SIGN
1	4/5/24	Matrix multiplication	1	
2	15/5/24	SCFS, SJF	5	
3		Round ~~Robin~~ robin & priority scheduling	11	
4	5/6/24	FCFS scheduling for ~~exchange~~ each queue ~~theory~~	19	10
5	5/6/24	Rate Monotonic & Earliest-deadline first	23	
6	12/6/24	Product consumer problem using semaphores	30	
8	12/6/24	C Program to simulate Dining Philosopher problem	33	10
9	19/6/24	C Program to simulate Bankers algorithm	37	
10	19/6/24	C Program to simulate deadlock detection	41	10
11	03/07/24	C Program to simulate contagious memory allocation technique	45	
12	03/07/24	C Program to simulate page replacement algo	50	10
13	10/07/24	C Program to simulate disk scheduling algo a) FCFS b) SCAN c) C-SCAN	57	10

M T W T F S S
Page No.: 1
Date:
YOUVA

Matrix calculations

```
# include <stdio.h>
void input ( int mat [][10], int row, int col ) {
    printf ("enter element");
    for ( int i = 0; i < row; i++ )
        for ( int j = 0; j < col; j++)
            scanf ("%d", & mat [i] [j]);
        }
    }
}

void add ( int mat [][10], int mat2[][10], int result [][10]
{
    for ( int i = 0; i < row; i++ ) {
        for ( int j = 0; j < col; j++ ) {
            result [i] [j] = mat1 [i] [j] +
                             mat2 [i] [j];
        }
    }
}

void sub ( int mat1 [][10], int mat2[][10], int result[]
{
    for ( int i = 0; i < row; i++ ) {
        for ( int j = 0; j < col; j++) {
            result [i] [j] = mat1 [i] [j] - mat [i
        }
    }
}

void multi ( int mat1 [][10], int mat2 [][10], int
             result, int row, int col ) {
    for ( int i = 0; i < row1; i++ )
        for ( int j = 0; j < col2; j++) {
        result (i) [j] = 0
```

```
        for (int k = 0; k < col1; k++) {
            result[i][j] += mat1[i][k] * mat2[k][j];
        }
      }
    }
  }

void display (int mat[][10], int row, int col)
{
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
      printf ("%d", mat[i][j]);
    }
    printf ("\n");
  }
}

int main ()
{
int mat1[10][10], mat2[10][10], result[10][10];
int row1, col1, row2, col2, choice;

printf ("no of rows");
scanf ("%d %d", &row1, &row2);
input (mat2, row2, col1);

printf ("no of rows");
scanf ("%d %d", &row2, &col2);
input (mat2, row2, col2);

printf ("choice");
scanf ("%d", &choice);
```

```
switch (choice) {
    case 1 :
        if (row1 = row2 && col1 = col2) {
            add (mat1, mat2, result, row1, col1)
            display (result, row1, col1);
        }
        else {
            printf (" not possible "); }
        break;

    case 2 :
        if (row1 == row2 && col1 == col2) {
            sub (mat1, mat2, result, row1, col1);
            display (result, row1, col);
        }
        else {
            printf (" not possible ");
        }
        break;

    case 3 :
        if (col1 == row2)
            multi (mat1, mat2, result, row1, col
                col2);
            display (result, row1, col2);
        }
        else {
            printf (" not possible ");
        }
        break;
    default :
        print (" invalid ");
    }
```

```
return 0;
}
```

Output:

```
Enter the rows and col of matrix 2 2

Enter the first matrix 1
2
3
4

Enter the second matrix 2
3
4
5
6 7

Sum of the two matrix is
3  5
6  11
```

Neha 8/5/24

Lab 2

Q) Write a C program to simulate the following non-pre-emptive CPU scheduling algo to find turnaround time and waiting time.
a) FCFS
b) SJF

```
#include <stdio.h>
int n, i, j, pos, temp, choice, Burst_time[20],
waiting_time[20], turn_around_time[20], process[20],
total = 0;

float avg_turn_around_time = 0;
avg_waiting_time = 0;

int FCFS ()
{
  waiting_time[0] = 0;
  for (i = 1; i < n; i++)
    {
      waiting_time[i] = 0;
      for (j = 0; j < i; j++)
        waiting_time[i] += Burst_time[j];
    }

printf ("\n Process \t\t Burst time \t\t waiting
        time \t\t turnaround time");

for (i = 0; i < n; i++)
{
  turn_around_time[i] = Burst_time[i] + waiting_time
```

```
printf("\n P[%d] \t %d \t %d \t %d \t %d", i+1,
        Burst_time[i], waiting_time[i], turn_
        around_time[i]);
}

avg_waiting_time = (float)(avg_waiting_time)/
                   (float)i;

avg_turn_around_time = (float)(avg_turn_time)/
                       (float)i;

printf("\n Avg wait time : %.2f", avg_waiting_time);

printf("\n Average turnaround time : %.2f \n", avg_turn_around);
return 0;
}
int SJF()
{
 for(i=0; i<n; i++)
  {
   pos = i;
   for(j++; j<n; j++)
   {
    if(Burst_time[j] = Burst_time[pos])
      pos = j;
    }
   temp = Burst_time[i]
   Burst_time[i] = Burst_time[Pos];
   Burst_time[pos] = temp;
}
waiting_time[0] = 0;
for(i=1; i<n+i++)
   {
```

```
waiting_time[i] = 0;
for(j=0; j<i; j++)
    waiting_time[i] += Burst_time[j];
total += waiting_time[i];
}

avg_waiting_time = (float)total/n;
total = 0;
printf("\n Process \t\t Burst time \t\t waiting
        time \t\t turnaround time")

for(i=0; i<n; i++)
{
  turn_around_time[i] = Burst_time[i] + waiting_
                        time[i];
  total += turn_around_time[i];
  printf("\n P[%d] \t\t %d \t\t %d \t\t %d", process[i],
        Burst_time[i], waiting_time[i], turn_around
        _time[i]);
}

avg_turn_around_time = (float)total/n;
printf("\n Avg wait time = %f", avg_waiting_time);

printf("\n avg turnaround time = %f\n", avg_turn_around
        time);

int main()
{
  printf("Enter the total number of processes:");
  scanf("%d", &n);
  printf("\n enter burst time:\n");
  for(i=0; i<n; i++)
  {
```

```
{
printf("P[%d];", i+1);
scanf("%d", &Burst_time[i]);
process[i] = i+1;
}

while(1)
{
printf("\n_Main_Menu_\n");
printf("\n1. FCFS scheduling \n2. SJF
        scheduling \n");

    printf("\n Enter your choice ");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1: FCFS();
              break;

      case 2: SJF();
              break;

      default: printf("Invalid");
    }
  }
return 0;
}
```

output

Enter the total no. of processes : 5
Enter Burst time :
P[1] : 20
P[2] : 5
P[3] : 30
P[4] : 15
P[5] : 8

-- Main Menu --
1. FCFS scheduling
2. SJF scheduling
Enter your choice : 1

Process	Burst time	waiting time	turnaround time
P[1]	20	0	20
P[2]	5	20	25
P[3]	30	25	55
P[4]	15	55	70
P[5]	8	70	78

Average waiting time = 34.00
Average turnaround time = 44.60

-- Main Menu --
1) FCFS scheduling
2) SJF scheduling
Enter your choice : 2

Process	Burst time	waiting time	Turnaround time
P[2]	5	0	5
P[5]	8	5	13
P[4]	15	13	28
P[1]	20	28	48
P[3]	30	48	78

Average waiting time = 18.794
Average turnaround time = 34.4002

Lab 3

Write a c program to simulate the following CPU scheduling algo to find turnaround time and waiting time.

a) Priority (pre-emptive & non-pre-emptive)
b) Round Robin (experiment with different quantum size for RR algo.

Sol:

Priority (Non-pre-emptive)

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

struct process {
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void find_average_time ( struct process [], int);
void priority - scheduling ( struct process [], int);

int main ()
{
    int n, i;
    struct process proc [10];
    printf ("Enter the no. of process : ");
    scanf ("%d", &n);
```

```
for (i = 0; i < n; i++)
  {
   printf ("\n enter the process id : ");
   scanf ("%d", &proc[i].process_id);

   printf ("enter the burst time : ");
   scanf ("%d", &proc[i].burst_time);

   printf ("\n enter the priority ");
   scanf ("%d", &proc[i].priority);
  }
  priority_scheduling (proc, n);
  return 0;
}

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

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

```
void find_avg_time ( struct process proc [], int n)
{
 int wt [10], tat [10], total_wt = 0, total_tat = 0;

 find_waiting_time ( proc, n, wt, tat );

 for ( j = 0; j < n; j++)
 {
 total_wt = total_wt + wt [j];
 total_tat = total_tat + tat [j];
 }
 printf ( "%f", (float) total_wt / n);
 printf ( "%f", (float) total_tat / n);
 }

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

Output :-

Enter no of process : 5
Enter process ID = 1
Enter burst time = 4
Enter priority = 2

PID = 2
BT = 3
P = 3

PID = 3
BT = 1
P = 4

PID = 4
BT = 5
P = 5

PID	BT	P	CMT	TAT
1	4	2	6	4
2	3	3	4	7
3	1	4	7	8
4	5	5	1	13
5	2	5	13	25

Avg WT = 0.4
Avg TAT = 1.4

b) Round Robin (Non- pre- emptive)

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

int turnaroundtime ( int processes [], int n, int bt[],
 int wt [], int tat[] ){
    for ( int i = 0; i < n; i++ )
    tat [i] = bt [i] + wt [i];
    return 1;
  }


int waitingtime (int processes [], int n, int bt [], int wt [],
 int quantum )
{
 int rem_bt [n];
 for ( int i = 0; i < n; i ++ )
 rem_bt [i] = bt [i]
 int t = 0;
 while (1)
 {
   bool done = true;
   for ( int i = 0; i < n; i++ )
   {
    if ( rem_bt [i] > 0 )
    {

     done = false;
     if ( rem_bt [i] > quantum )
      {
        t + = quantum;
        rem_bt [i] -= quantum;
      }

      else {
```

```
            t = t + rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
}
if (done == true)
    break;
}
return 1;
}

int findavgTime (int process[], int n, int bt[]) {
    int wt[n], tat[n], total_wt = 0;

    waitingtime (processes, n, bt, wt, quantum);
    turnaroundtime (processes, n, bt, wt, tat);
    printf ("\n\n Processes, n, bt, wt, tat);
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
    }
printf ("\n Average waiting time = %.d", total_wt)
}
}

int main (
{
    int n, processes[n], burst_time[n];
    quantum;
    printf ("Enter the no of processes");
    scanf ("%d", &n);
```

```
printf ("\n Enter the no of quantum time : ");
scanf ("%d", &quantum);
int i = 0
for (i=0; i<n; i++)
{
  printf ("\n Enter the process : ");
  scanf ("%d", &process[i]);
  printf ("Enter the burst time");
  scanf ("%d", &bt[i]);
}
findavg_time (process, n, burst_time, quantum);
    return 0;
}
```

Output :-

Enter no of process = 2
Enter quantum time = 2
Enter process & burst time
2 1
4 3

Process	Burst time	waiting	Turnaround time
1	1	0	1
2	3	1	4

Average waiting time = 0.5
Average turnaround time = 2.5

Lab-4

Write a C program to simulate multi-level queue scheduling algo considering the following scenario.

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

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void FCFS ( struct process * queue, int n ) {
    int i, j;
    struct process temp;
    for ( i=0; i<n; i++ ) {
        for ( j= i+1 ; j<n; j++ ) {
            if ( queue [i] . arrival_time > queue [j] .
                 arrival_time ) {
                temp = queue [i];
                queue [i] = queue [j];
                queue [j] = temp;
            }
        }
    }
}

int main ( ) {
    int n, i;
```

```
struct process * system_queue, * user_queue;
int system_n = 0, user_n = 0;
float avg_waiting_time = 0, avg_turnaround_time = 0;

printf("Enter the number of processes: ");
scanf("%d", &n);

system_queue = (struct process *) malloc(n * sizeof(
struct process));
user_queue = (struct process *) malloc(n * sizeof
(struct process));

for(i = 0; i < n; i++){
    struct process p;
    printf("Enter arrival time, burst time,
    and priority");
    scanf("%d %d %d", &p.arrival_time, &p.burst
    _time, &p.priority);
    p.pid = i+1;
    p.waiting_time = 0;
    if(p.priority == 0){
        system_queue[system_n++] = p;
    } else {
        user_queue[user_n++] = p;
    }
}

FCFS(system_queue, system_n);
FCFS(user_queue, user_n);
int time = 0;
int s = 0, u = 0;
while(s < system_n || u < user_n){
    if(system_queue[s].arrival_time <=
```

```
time ) {
    if ( user_queue [0]. arrival_time <= time && user
    queue [u ]. arrival_time < system_queue [s]. arrival_
    time < system_queue [s]. arrival)
    }
    else {
        system_queue [s]. waiting_time = time - system_
        queue [s]. arrival_time ;
        time += system_queue [s]. burst_time ;
        system_queue [s]. turnaround_time =
        system_queue [s]. waiting_time + system_queue [s].
        burst_time ;
    }
}
else if ( user_queue [u]. arrival_time <= time ) {
    user_queue [u]. waiting_time = time - user_queue [u]
    avg_waiting_time += user_queue [u]. waiting_time ;
    u++;
}
else {
    time = user_queue [u]. arrival_time ;
}
}
}

avg_waiting_time /= n;
avg_turnaround_time /= n;

for ( i=0; i< system_n; i++ )
    system_queue [i]. burst_time , system_queue [i]
    fritons )
    system_queue [i]. waiting_time, system_queue [i].
    turnaround_time );
```

```
for (i = 0; i < usr_n; i++){
    usr_queue[i].burst_time, usr_queue[i].
    priority,
    usr_queue[i].waiting_time, usr_queue[i].
    turnaround_time);
}

printf("Average waiting time %.2f\n", avg_waiting_time
printf("Average turnaround time: %.2f\n", avg_tat_time

free(system_queue);
free(usr_queue);
return 0;
}
```

Output :-

```
Enter the no. of process : 2
Enter arrival time, burst time, and priority : 2
                                               5
                                               3
Enter arrival time, burst time and priority  6
                                             5
                                             12
```

PID	Burst time	Priority	queue	w +	TAT
1	5	3	queue	0	5
2	5	12	queue	1	6

```
Average WT : 0.50
Average Turnaround time : 5.50
```

* C Program for
a) Rate - monotonic
b) Earliest - deadline first

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

# define Max_Process 10

typedef struct {
    int id;
    int burst_time;
    float priority;
} Task;

int num_of_process;
int execution_time[Max_Process], period[Max_Process],
remain_time[Max_Process], deadline[Max_Process];

void get_process_info (int selected_algo)
{
  printf ("Enter total no of process");
  scanf ("%d", &no_of_process);
  if (num_of_process < 1)
    {
      exit 0;
    }

for (int i = 0; i < num_of_process; i++)
  {
    printf ("==> Execution time :")
    scanf ("%d", &execution_time[i]);
```

```
remain_time[i] = execution_time[i];
if (selected_algo == 2)
{
printf("Enter deadline: ");
scanf("%d", &period[i]);
}
}
}

int max(int a, int b, int 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_observation_time(int selected_algo)
{
if (selected_algo == 1)
{
return max(period[0], period[1], period[2]);
}
else if (selected_algo == 2)
{
return max(deadline[0], deadline[1], deadline[2]);
}
}
```

```
void print_schedule ( int process_list [], int cycles)
{
  printf ("\n scheduling : \n")
  printf ("  time ");
  for ( int j = 0; j < cycles; j++ )
  {
    if (j < 10)
      printf ( "| 0 %d ", j );
    else
      printf ( "| %d", j );
    }
  printf ("| \n");
  for ( int j = 0; j < num_of_process; j++ )
  {
    printf ("P [%d] :", j+1);
    for ( int g = 0; g < cycles; g++ )
    {
      if ( process_list [g] == j + 1 )
        printf ("  --  ");
      else
        printf ("|   ");
      }
    printf ("| \n");
  }
}

void rate_monotonic ( int time )
{
  int process_list [100] = 204, min = 999, next
  next_process = 0;
  float utilization = 0;
  for ( int i = 0; i < num_of_process; i++)
  {
```

```
        utilization += (1.0 * execution_time[i]) / period[i];
    }
    int n = num_of_process;
    int m = (float)(n * (pow(2, 1.0/n) - 1));
    if (utilization > m)
    {
        printf(...)
    }
    for (int i = 0; i < time; i++)
    {
        min = 1000;
        for (int j = 0; j < num_of_process; j++)
        {
            if (remain_time[j] > 0)
            {
                if (min > period[j])
                {
                    min = period[j];
                    rest = period[j];
                }
            }
        }
    }
print_schedule (process_list, time);

void earliest_deadline_first (int time)
{
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++)
    {
        utilization += (1.0 * execution_time);
    }
    int n = num_of_process;
    int process[no_of_process];
    int max_deadline;
```

bool is_ready [num_of_process];

for (int i=0; i< num_of_process ; i++){
    if (deadline [i] > max_deadline)
        max_deadline = deadline [i];
    }

for (int i = 0; i< num_of_process ; i++)
    for (int j = i+1; j < num_of_process; j++){
        if (deadline [j] < deadline [i]){
            int temp = execution_time [j]
            execution_time [j] = execution_time [i];
            execution_time [i] = temp ;
            temp = deadline [j]
            deadline [j]= temp ;
            temp = process [j]
            process [j] = process [i];
            process [i] = temp;
        }
    }
}

for (int i = 0 ; i < num_of_process ; i++){
    remain_time [i] = execution_time [i];
    remain_deadline [i] = deadline [i];
}

for (int j = 0 ; j < num_of_process ; i++){
    -- deadline [i];
    if ((execution_time [i] == 0) && is_ready [i]){
        deadline [i] += remain_deadline [i];
        is_ready [i] = false;
    }

```
min_deadline = max_deadline;

current_process = -1;
for (int i = 0; i < num_of_process; i++)
{
    if ((deadline[i] <= min_deadline) && (
        execution_time[i] > 0))
    {
        current_process = i;
        min_deadline = deadline[i];
    }
}
print_schedule(process_list, time);
}
int main()
{
    int option;
    int observation_time;
    while(1)
    {
    printf("\n 1 Rate Monotonic \n 2. Earliest deadline first
    scanf("%d", &option);
    switch(option){
        case 1: get_process_info(option);
                observation_time = get_observation_time(
                rate_monotonic(observation_time);
                break;
        case 2: get_process_info(option);
                observation_time = get_observation_time(
                earliest_deadline_first(observation_time);
                break;
        case 3: exit(0);
                default: printf("error");
            }
        }
```

```
}
return 0;
}
```

Output

1. Rate Monotonic
2. Earliest Deadline first
3. Proportional scheduling

Enter your choice :- 1

Enter total no of process (max 10) : 2

Process 1 :-
execution time : 2
period : 3

Process 2 :-
execution time : 5
Period 62 : 6

Scheduling

Time	00	01	02	03	04	05
P[1] :	###	###		###	###	
P[2] :			###			###

5/6/24

* Write a C program to simulate producer - consumer problem using semaphores.

```
#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 \n 2. consumer \n 3 Exit ");
  while (1)
  {
  printf ("\n Enter 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: wait(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 ("\n Producer produces the item %d", x);
    mutex = signal (mutex);
}

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

Output :-

Note 1. Producer
2. Consumer
3. Exit
Enter your choice : 1

Producer produces the item 1
Enter your choice : 2

Producer produces the item 2
Enter your choice : 1

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

Consumer consumes item : 3
Enter your choice : 2

Consumer consumes item 2
Enter your choice : 2

Consumer consumes item 1
Enter your choice : 2
Buffer is empty !!

* Write a C program to simulate the concept of dining - Philosophers problem.

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

#define NS
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (j+4)%N
#define RIGHT (j+1)%N

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

sem_t mutex;
sem_t S[N];

void test (int j)
{
 if (state[j] == HUNGRY && state[LEFT] != EATI
  && state[RIGHT] != EATING)
  {
   state[j] = EATING
   sleep(2);
   printf("Philosopher %d takes fork %d and
   %d \n", j+1, LEFT +1, j+1);
   sem_post(&S[j]);
  }
}
```

```
void test (int

void take_fork (int i)
{
  sem_wait (&mutex);
  state[i] = Hungary;
  printf (" Philosopher %d is Hungary \n", i+1);
  test (i);

  sem_post (&mutex);
  sem_wait (&s[i]);
  sleep (1);
}

void put_fork(int i)
{
  sem_wait (&mutex);
  state[i] = THINKING;
  printf (" Philosopher %d putting fork %d 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++)
        sem_init(&s[i], 0, 0);
    for(i=0; i<N; i++)
    {
        pthread_create(&thread_id[i], NULL, philosopher, &ph
        printf("Philosopher %d is thinking \n", i+1);
    }
    for(i=0; i<N; i++)
    {
        pthread_join(thread_id[i], NULL);
    }
}
```

Output

Phi... 1 is thinking
Phil.. 2 is thinking
Phil.. 3 is thinking
Phil.. 4 is thinking
Phil.. 5 is thinking
Phil.. 1 is hungary
Phil.. 2 is hungary
Phil..3 is hungary

phil.. 4 is hungary
phil.. 5 is hungary
phil..1 takes fork 1 and 5
phil 1 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 4 is putting 3 and 4 down
phil.. 4 is thinking
phil 5 is putting 5 and 6 down
phil 5 is thinking

12/6/24

Program to simulate Bankers algo for the purpose of deadlock avoidance

```
#include <stdio.h>
int main()
{
int n, m, i, j, k
printf("enter the no of process");
scanf("%d", &n);
printf("Enter the no. of resources");
scanf("%d", &m);

int allocation[n][m];
printf("Enter the Allocation Matrix : \n");
for(i=0; i<n; i++)
{
  for(j=0; j<m; j++)
  {
    scanf("%d", &allocation[i][j]);
  }
}

int max[n][m];
printf("Enter the Max Matrix : \n");
for(i=0; i<n; i++)
  {
    for(j=0; j<m; j++)
    {
      scanf("%d", &max[i][j]);
    }
  }

int available[m];
printf("Enter the available resources");
for(i=0; i<m; i++)
```

```
{
  scanf ("%d", &available[i]);
}
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] - allocation[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] > available[j])
        {
          flag = 1;
          break;
        }
```

```
if ( flag == 0 )
{
 ans [ ind++ ] = i;
 for (y=0; y<m; y++)
 {
  available [y] += allocation [i] [y];
 }
 f [i] = 1;
}
}
}
}
int flag = 1;
for (j = 0; j<n; j++)
{
 if (f[j]==0)
 {
  flag = 0;
  printf ("The following system is not safe \n");
  break;
 }
}
if ( flag == 1)
{
 printf ("following is the SAFE Sequence \n");
 for (j = 0; j < n-1; j++)
 {
  printf ("P%d ->", ans[j]);
 }
  printf ("P%d \n", ans[n-1]);
 }
 return 0; }
```

output

Enter the no of processes : 5
Enter the no of resources : 3
Enter the allocation matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the max matrix :
3 3 2
7 5 3
9 0 2
2 2 2
4 3 3
Enter the available resources:
3 3 2
Following is the SAFE sequence
P0 → P1 → P2 → P3 → P4

* Write a C program for deadlock detection

```
#include <stdio.h>
static int mark[20];
int i, j, np, nr;

int main()
{
int alloc[10][10], request[10][10], avail[10], r[10], w[10]

printf("\n enter the no. of trucks:");
scanf("%d", &np);
printf("\n enter the no of resources");
scanf("%d", &nr);
for(i=0; i<nr; i++)
{
printf("\n total amount of the resource R%d:", i+
scanf("%d", &r[i]);
}

printf("\n enter the request matrix");

for(i=0; i<np; i++)
for(j=0; j<nr; j++)
scanf("%d", &request[i][j]);

printf("\n 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 ( j = 0; j < nf; j++)
        {
        avai [j] -= allot [i] [j];
        }
    }

    for (i = 0; i < nf; i++)
    {
    int count = 0;
    for (j = 0; j < nr; j++)
    {
      if ( allot [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 < nf; i++)
    {
     int canbeprocessed = 0;
      if ( mark [i] != 1)
    {
     for ( j = 0 ; j < nr ; j++)
     {
      if ( request [i] [j] <= w [j] )
          canbeprocessed = 1;
      else
```

```
    }
    canbeprocessed = 0;
    break;
   }
  }
 if ( canbeprocessed )
 {
  mark [i] = 1;
  for (j = 0; j < nr; 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 ("\n no deadlock possible");
}
```

Output

Enter the number of processes : 4
Enter the number of resources : 3
Total amount of the resource R1 : 10
Total amount of the resource R2 : 5
Total amount of the resource R3 : 7

Enter the required matrix :

2 0 0

3 0 2

2 1 1

Enter the allocation matrix :

0 1 0

2 0 0

3 0 2

2 1 1

No deadlock possible

19/6/24

* Write a C program to simulate the following contiguous memory allocation techniques:

a) Worst fit
b) Best fit
c) first-fit

a)

```
# include <stdio.h>
# define max 25
void firstFit (int b[], int nb, int f[], int nf);
void worstFit (int b[], int nb, int f[], int nf);
void bestFit (int b[], int nb, int f[], int nf)

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

```
printf ("\n memory management scheme - First fit");
firstFit (b, nb, f, nf);
printf ("\n\n Memory Management scheme - Worst fit");
worstF (b, nb, f, nf);
printf ("\n\n Memory Management scheme - Best fit");
return 0;
}
void firstfit (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 ("\n file_no \t file size : \t Block_no : \t Block");
void worstFit (int b[], int nb, int f[], int nf)
{
 int bf[max] = {0};
 int ff[max] = {0};
 int frag[max], j, i, temp, highest = 0;
```

```
for (i = 1; i <= nb; i++)
{
  for (j = 1; j <= nb; j++)
  {
   if (bb[j] != 1)
   {
    temp = b[j] - b[i];
    if (temp >= 0 && highest < temp)
    {
     bb[i] = j;
     highest = temp;
    }
   }
  }

 frag[i] = highest;
 bb[bb[i]] = 1;
 highest = 0;
}

void lastFit (int b, int nb, int b[], int nb)
{
 int bb[max] = {0};
 int bb[max] = {0};
 int frag[max] = 0, i, j, temp, lowest = 10000;

 for (i = 1; i <= nb; i++)
  {
   for (j = 1; j <= nb; j++)
   {
    if (bb[j] != 1)
    {
     temp = b[j] - b[i];
     if (temp >= 0 && lowest > temp)
```

```
        {
        bf[j] = i;
        lowest = temp;
        }
    }
}
frag[i] = lowest;
bf[bf[i]] = 1;
lowest = 10000;
}
printf("\n file_no:\t file_size:\t Block\t block_size");
{
    printf("\n %d \t\t %d \t\t %d \t\t %d", i, f[i],
            bf[i], b[bf[i]], frag[i]);
    }
}
```

Output

Enter the no of blocks: 2
Enter the no of files: 2

Enter the size of the blocks:
Block 1: 50
block 2: 60

Enter the size of the files
File 1: 24
File 2: 56

Date:

First fit

Fil_no	Fil_size	Block_no	Block_size	Fragment
1	24	0	50	26
2	36	1	60	0

Memory Management schem - Worst fit

Fil no	Fil size	Block_no	Block_size	Fragment
1	24	1	60	36
2	60	0	50	0

* Write a C program to simulate page replacement algo

a) FIFO
b) LRU
c) Optimal

```
# include <stdio.h>
int n, b, i, j, k;
int in[100];
int p[50];
int hit = 0;
int pgfaultcnt = 0;

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

void initialize ()
{
pgfaultcnt = 0;
for (i = 0; i < b; i++)
p[i] = 9999;
}
int isHit (int data)
{
    hit = 1;
```

```
        break;
      }
    }
    return hit;
}
int gethitindex (int data)
{
    int hitind;
    for (k=0; k< f; k++)
    {
      if (f[k] == data)
      {
        hitind = k;
        break;
      }
    }
    return hitind;
}
void dispPages ()
{
    for (k=0; k< f; k++)
    {
      if (f[k] != 9999)
        printf (" %d", f[k]);
      }
    }

void dispPgFaultCnt()
{
    printf ("\n total no of page faults: %d", pgfaultcn
}
```

```
void fifo()
{
getdata();
initialize();
for(i=0; i<n; i++)
{
printf("\n for %d :", in[i]);
if(isHit(in[i])==0)
{
for(k=0; k<f-1; k++)
    h[k] = h[k+1]

h[k] = in[i]
hg faultcnt++;
dispPages();
}
else
  printf("No page fault");
}
dispHgFaultCnt();
}

void optimal()
{
initialize();
int near[50];
for(i=0; i<n; i++)
{
printf("\n for %d :", in[i]);
if(isHit(in[i])==0)
{
for(j=0; j<k; j++)
{
```

```
int tg = h[j]
int bound = 0;
for (k=i; k<n; k++)
{
  if ( tg == in[k])
  {
    near[j] = k;
    bound = 1;
    break = ;
  }
  else
    bound = 0;
  }
  if (! bound)
    near[j] = 9999;
 }
int max = -9999;
int nxtindex;
for (j=0; j<n-1; j++)
{
  if ( near[j] < max)
{
  max = near[j];
  nxtindex = j;
 }
}
h[nxtindex] = in[i];
tg bouttint ++;
display h tags();
}
  else
    printf ("No tags found");
  }
```

```
disp Page Fault Cnt ()
{
void run ()
{
initialize ();
int hast [50];
for ( j=0; j<n; j++)
{
printf ("In for %d : " in [i]);
if ( is Hit ( in [i] ) == 0 )
{
 for ( j = 0; j<nf ; j++)
 {
  int pg = f [j] ==k;
  int found = 0;
  for ( k=i-1; k>=0; k--)
  {
   ~~int~~ if (pg == in [k])
   {
    hast [j ] = k;
    found = 1;
    break;
   }
  else
   found = 0;
  }
if (! found )
   hast [j ] = - 9999;
 }
int min = 9999;
int refindex ;
for ( j=0; j<nf ; j++)
{
```

```
    if ( least [i] < min )
    {
      min = least [i]
      replinden = i;
    }
  }
t [ replinden ] = in [i];
hg faultent ++;
disp lages ();
else ();
    printf ("No page fault !");
  }
  disp fg Fault(nt ();
}
int main ()
{
 int choice;
 while (1)
 {
  printf ("In Page Replacement Algo In 2 Enter data In 2
  FIFO In 3. optimal In 4. LRU In 5 Exit In
  Enter 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 Algo
1) Enter data
2) FIFO
3) Optimal
4) LRU
5) Exit
Enter your choice 2
Enter length of page reference sequence :-12
Enter the page reference sequence : 8 1 2 3 4 1 2 3 4 5 8 8

Enter no of Frames : 2
Fault 1 : 8
Fault 2 : 8 1
Fault 3 : 1 2
Fault 4 : 2 3
Fault 5 : 3 4

Enter no of frames 3
Ref 1 : 1
Ref 2 : 1 2
Ref 3 : 1 2 3
Ref 4 : 2 3 4
Ref 1 : 3 4 1
Ref 2 : 4 1 2
Ref 5 : 1 2 5
Ref 1 : No page fault
Ref 9 : no page fault
Ref 3 : 9 5 3
Ref 4 : 5 3 4
Ref 5 : No page fault
Total no of page fault : 9

optimal
Ref 1 : 1
Ref 2 : 12
Ref 3 : 1 2 3
Ref 4 : 1 2 3 4
Ref 1 : no page fault
Ref 2 : no page fault
Ref 5 : 1 2 5
Ref 1 : No page fault
Ref 2 : No page fault
Ref 3 : No page fault
Ref 4 : 4 2 5
Ref 5 : no page fault
Total no of page fault : 7

Sneha
3/7/24

* Write a C program to simulate disk scheduling algo

a) FCFS
b) SCAN
c) C-SCAN

a)
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int RQ[100], i, n, totalheadMoment = 0, initial;
printf("Enter the no of Request \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 \n");
scanf("%d", &initial);

for (i = 0; i < n; i++)
{
totalheadMoment = TotalHeadMoment + abs(RQ[i] - initial);
initial = RQ[i];
}
printf("Total head moments is %d", totalhead
return 0;
}
```

output:
```
Enter the no. of request : 5
Enter the Request sequence :
98 183 37 122 14
Enter initial head position : 53
Total head movement is 469
```

1)

```
# include < stdio.h >
# include < stdlib.h >
int main ()
{
int RQ [100], i, j, n, TotalHeadMoment = 0, initial, size, move;
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 \n");
scanf ("%d", &initial);
printf ("Enter ~~initial~~ total disk size \n");
scanf ("%d", &size);
printf ("Enter the head movement direction for high 1

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

```
        break;
      }
    }

  if ( move == 1)
    {
      for ( i = index ; i < n; i++)
        {
          totalheadmovement = Totalheadmovement + abs ( RQ[i] - initial);
          initial = RQ[i];
        }

  Total Head Movement = Totalhead Movement + abs ( size - RQ [i-1] - 1);
  initial = size - 1;
  for ( i = index - 1; i >= 0 ; i-- )
    {
      total head movement = Total Head Movement + abs ( RQ [i] - initial);
      initial = RQ [i];
    }
  }

else {
    for ( j = index - 1; j >= 0; j-- )
      {
        Total Head Movement = Total Head Movement + abs ( RQ [i] - initial);
        initial = RQ [i];
      }
  Total Head Movement = Total Head Movement + abs ( RQ [i+1] - 0);
  initial = 0;
  for ( i = index; i < n; i++ )
  {
    Total headmovement = Total head Movement + abs ( RQ [i] - initial);
    initial = RQ [i];
```

```
    }
}
printf ("Total head movement is %d", TotalHeadMoment);
return 0;
}
```

Output :-

```
Enter the no of request : 8
Enter the Request sequence :
98  183  37  122  14  124  65  67
Enter initial head position : 53
Enter total disk size : 200
Enter the head movement direction : 1
Total head movement is 363
```

c)

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

for ( i = 0; i < n; i++)
scanf ("%d", &RQ[i] );
printf ("Enter initial head position \n")
scanf ("%d", initial );
printf ("%d", &size );

for (i=0; j<n-1 ; j++)
  {
    for ( RQ [i] > 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 ( initial < RQ [i])
    {
      index = i ;
```

```
        break;
      }
    }
    if ( num == 1 )
    {
      TotalHeadMoment = TotalHeadMoment + abs( RQ[i] - initial);
      initial = RQ[i];
    }
  TotalHeadMoment = TotalHeadMoment + abs ( size - RQ[i-1] - 1 );
  TotalHeadMoment = TotalHeadMoment + abs ( size - 1 - 0 );
  initial = 0;
  for ( i=0 ; i< index ; i++ )
  {
    totalheadmoment = totalheadmoment + abs (RQ[i] - initial);
    initial = RQ[i];
  }
}

else
{
  for ( i= index -1 ; i >= 0; i--)
  {
  totalheadmoment = totalheadmoment + abs( RQ[i] - initial);
    initial = RQ[i];
  }
totalheadmoment = TotalHeadMoment + abs (RQ[i+1] - 0];
totalheadmoment = TotalHeadMoment + abs ( size - 1 - 0);
initial = size - 1;
for( i = n-1; i >= index; i--)
{
  totalheadmoment = totalHeadMoment + abs (RQ[i] -
  initial);
  initial = RQ[i];
}
```

```
}
printf ("Total head moment is %d", total head
  moment);
return 0;
}
```

Output :-

```
Enter the no of request : 4
Enter the request sequence :
 32
 45
 50
 82
Enter the initial head position : 12
Enter the total disk size :- 30
Enter the head moment direction :- 5
Total head moment is : 168
```

Viva
10/7/24