OS observation book

INDEX

NAME: Agneya D A STD.: SEC.: 4-A ROLL NO.: 024

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	8/5/24	Matrix addition, subtraction, multiplication	1-4	10
2.	15/5/24	FCFS & SJF scheduling	5-8	
3.	22/5/24	Priority & round robin scheduling	9-15	9
4	5/6/24	Rate-monotonic & earliest deadline first	16-23	
5	19/6/24	Producer-consumer, Philosophers dining	24-29	10
6	19/6/24	Deadlock detection & avoidance	30-35	10
7	3/7/24	Contiguous memory allocation & page memory allocation.	36-46	10
8	10/7/24	FCFS, SCAN, c-SCAN Disk scheduling	47-53	10

Date 8/5/2024
Page 1

Matrix Addition, Subtraction, Multiplication

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int a[MAX][MAX], b[MAX][MAX], addM[MAX][MAX];
int subM[MAX][MAX], mulM[MAX][MAX], m,n;
void add();
void subtract();
void multiply();
void main(){
   int i,j;
   printf("Enter the value of m and n: ");
   scanf("%d%d",&m,&n);
   printf("Enter the matrix A: ");
   for(i=0; i<m; i++){
      for(j=0;j<n;j++){
         scanf("%d",&a[i][j]);
      }
   }
   printf("Enter the matrix B: ");
   for(i=0; i<m;i++){
     for(j=0; j<n;j++){
        scanf("%d",&b[i][j]);
     }
   }
   add();
   subtract();
   multiply();
}

void add(){
   int i,j;
```

```
for(i=0; i<m; i++){
    for(j=0; j<n; j++){
        addM[i][j] = a[i][j] + b[i][j];
    }
}
printf("A + B = \n");
for(i=0; i<m; i++){
    for(j=0; j<n; j++){
        printf("%d", addM[i][j]);
    }
    printf("\n");
}
}
void subtract(){
    int i,j;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            subM[i][j] = a[i][j] - b[i][j];
        }
    }
    printf("\n A - B = \n");
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            subM[i][j] = a[i][j] - b[i][j];
        }
    }
    printf("\nA - B = \n");
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            printf("%d", subM[i][j]);
        }
        printf("\n");
    }
}
```

```
void multiply(){
  int i,j,k;
  for(i=0; i<m; i++){
    for(j=0; j<n; j++){
      mulM[i][j]=0;
      for(k=0; k<n; k++){
        mulM[i][j] += a[i][k]*b[k][j];
      }
    }
  }
  printf("\n A * B = \n");
  for(i=0; i<m; i++){
    for(j=0; j<n; j++){
      printf("%d ", mulM[i][j]);
    }
    printf("\n");
  }
}
```

Output:

Enter the value of m and n: 2 2

Enter the matrix A: 1

2

3

4

Enter the matrix B: 1

1

1

1

Matrix A:

1 2

3 4

Matrix B:

1 1

1 1

A + B =

2 3

4 5

A - B =

0 1

2 3

A * B =

3 3

7 7

8/5/24

First Come First Serve and Shortest Job First CPU scheduling

Date 15/5/2024
Page 5

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

int FCFS(){
  WaitingTime[0]=0;
  for(i=0;i<n;i++){
    WaitingTime[i]=0;
    for(j=0;j<i;j++){
      WaitingTime[i]+=BurstTime[j];
    }
  }
  printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround
  Time");
  for(i=0;i<n;i++){
    TurnAroundTime[i]=BurstTime[i]+WaitingTime[i];
    ATAT  +=TurnAroundTime[i];
    AWT   +=WaitingTime[i];
    printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,
  BurstTime[i],WaitingTime[i],TurnAroundTime[i]);
  }
  AWT=(float)AWT/n;
  ATAT=(float)ATAT/n;
  printf("\nAverage Waiting Time:%.2f",AWT);
  printf("\nAverage Turnaround Time:%.2f\n",ATAT);
}

int SJF(){
  for(i=0;i<n;i++){
    pos=i;
    for(j=i+1;j<n;j++){
      if(BurstTime[j]<BurstTime[pos]){pos=j;}}
```

```
        temp = BurstTime[i];
        BurstTime[i] = BurstTime[pos];
        BurstTime[pos] = temp;
        temp = process[i];
        process[i] = process[pos];
        process[pos] = temp;
    }

    waitingTime[0] = 0;
    for(i=1; i<n; i++){
        for(j=0; j<i; j++)
            waitingTime[i] += BurstTime[j] + waiting ;
        total += waitingTime[i];
    }

    AWT = (float) total/n;
    total = 0;
    printf("\n Process\t\tBurst Time\t\tWaiting Time\t\t
    Turnaround Time");
    for(i=0; i<n; i++){
        TurnAroundTime[i] = BurstTime[i] + waitingTime[i];
        total += TurnAroundTime[i];
        printf("\n P[%d]\t\t\t%d\t\t\t%d\t\t\t%d", process[i],
        BurstTime[i], WaitingTime[i], TurnAroundTime[i]);
    }

    ATAT = (float) total/n;
    printf("\n\n Average Waiting Time = %f", AWT);
    printf("Average Turnaround Time = %f\n", ATAT);
}

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("\nEnter Burst Time:\n");
    for(i=0; i<n; i++){
        printf("P[%d]:", i+1);
        do{
            scanf("%d", &BurstTime[i]);
        }while(BurstTime[i]<0);
        process[i] = i+1;
    }
    while(1){
        printf("\n---- MAIN MENU ---- \n");
        printf("1. FCFS Scheduling \n2. SJF scheduling\n");
        printf("\nEnter your choice:");
        scanf("%d", &choice);
        switch(choice){
            case 1: FCFS();
                    break;
            case 2: SJF();
                    break;
            case 3: exit(0);
            default: printf("Invalid Input!!!");
        }
    }
    return 0;
}
```

Output:

```
Enter total number of processes: 5
Enter Burst Time:
P[1]: 10
P[2]: 1
P[3]: 2
P[4]: 1
```

P[5]:5

---- MAIN MENU---

1. FCFS scheduling

2. SJF scheduling

Enter your choice: 1

~~---- MAIN MENU---~~

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	10	0	10
P[2]	1	10	11
P[3]	2	11	13
P[4]	1	13	14
P[5]	5	14	19

Average Waiting Time: 9.600000

Average Turnaround Time: 13.400000

---- MAIN MENU---

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice: 2

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	1	0	1
P[4]	1	11	12
P[3]	2	13	15
P[5]	5	17	22
P[1]	10	23	33

Average Waiting Time: 12.800000

Average Turnaround Time: 16.600000

---- MAIN MENU---

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice: 3

+5 | 5 | 24

```
#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 number of processes: ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("\nEnter the process ID: ");
        scanf("%d", &proc[i].process_id);
        printf("Enter the burst time: ");
        scanf("%d", &proc[i].burst_time);
        printf("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 proc[], int
n , int wt[], int tat[]){
    int i;
    for(i=0; i<n; i++){
        tat[i]= proc[i].burst_time +wt[i];
    }
}
void find_average_time (struct process proc[] int n
{
int wt[10], tat[10], total_wt =0, total_tat =0, i;
find_waiting_time (proc, n, wt);
find_turnaround_time(proc, n, wt, tat);
printf("\n Process ID\tBurst Time\t Priority\t Waiting
Time\t Turnaround Time");
for (i=0; i<n; i++)
{
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d", proc[i].
    process_id, proc[i].burst_time, proc[i].priority, wt[i],
    tat[i];
}
```

```
printf("\n\n Average Waiting Time = %.f", (float) total_wt
/n);
printf("\n Average Turnaround Time = %.f\n", (float) total_tat
/n);

void priority_scheduling (struct process proc[], int n)
{
  int i, j, pos;
  struct process temp;
  for (i=0; i<n; i++)
  {
    pos = i;
    for (j=i+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 number of processes: 5
Enter the process ID: 1
Enter the burst time: 4
Enter the priority 2

Enter the process ID: 2
Enter the burst time: 3
```

Enter the priority: 3

Enter the process ID: 3
Enter the burst time: 1
Enter the priority: 4

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

Enter the process ID: 5
Enter the burst time: 2
Enter the priority: 5

Process ID	Burst Time	Priority	Waiting Time	Turnaround 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

Average waiting Time = 6.400000
Average Turnaround Time = 9.400000

b) Round Robin (NON-pre-emptive)

```
#include <stdio.h>
#include <stdbool.h>
int turnaround time (int processes [], int n, int bt[],
 int wt[], int tat [])
    for (int i=0; i<n; i++)
        tat[i] = bb[i] + wt[i];
        return 1;
```

```
}

int waitingtime(int processes[], 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] > quantum){
                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
quantum) {
```

```
int wt[n], tat[n], total_wt = 0, tat = 0;
waitingTime(processes, n, bt, wt, quantum);
turnaroundtime(processes, n, bt, wt, tat);
printf("\n\nProcesses \t\t Burst Time \t\t Waiting
Time \t\t turnaround time\n");
for(int i=0; i<n; i++)
{
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("\n\t %d \t\t\t %d \t\t\t %d\n", i+1,
    bt[i], wt[i], tat[i]);
}
printf("\n Average Waiting Time = %f", (float) total_wt
/(float)n);
printf("\n Average turnaround time = %f", (float) total_tat
/ (float)n);
return 1;
}
int main()
{
    int n, ~~processes[n], burst_time[n],~~ quantum;
    printf("Enter the Number of Processes: ");
    scanf("%d", &n);
    int processes[n], burst_time[n];
    printf("\n Enter the quantum time: ");
    scanf("%d", &quantum);
    int i=0;
    for(i=0; i<n; i++){
        printf("\nEnter the process");
        scanf("%d", &processes[i]);
        printf("Enter the Burst time : ");
        scanf("%d", &burst_time[i]);
    }
```

```
    findavgTime(processes, n, burst-time, quantum);
    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter the quantum time: 3
Enter process: 1
Enter Burst time : 4
Enter process : 2
Enter Burst Time: 3
Enter process : 3
Enter Burst Time : 1
Enter process : 4
Enter Burst Time : 5
Enter the Process : 5
Enter the Burst Time : 2
```

Processes	Burst Time	Waiting Time	Turnaround Time
1	4	9	13
2	3	3	6
3	1	6	7
4	5	10	15
5	2	10	12

```
Average Waiting time = 7.600000
Average turnaround time = 10.600000
```

Rate- monotonic & 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],
remain-deadline[MAX-PROCESS];
void get-process-info (int selected-algo){
    printf("Enter the total number of processes
    (maximum %d): ");
    scanf("%d", &num-of-process);
    if(num-of-process < 1){
        exit(0);
    }
    for(int i=0; i<num-of-process; i++){
        printf("\nProcess %d:\n", i+1);
        printf("  ==> Execution time: ");
        scanf("%d", &execution-time[i]);
        remain-time[i] = execution-time[i];
        if(selected-algo == 2){
            printf("==> Deadline: ");
            scanf("%d", &deadline[i]);
        }
        else{
            printf("==> Period: ");
```

```
        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;
    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\n");
    printf("Time: ");
    for (int i=0; i< cycles ; i++) {
        if (i<10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
```

```
        printf("|\n");
        for(int i=0; i<num_of_process; i++) {
            printf("P[%d]: ", i+1);
            for(int j=0; j<cycles; j++) {
                if (process_list[i]==i+1)
                    printf("####");
                else
                    printf("|  ");
            }
            printf("|\n");
        }
    }

void rate_monotonic(int time) {
    int process_list[1000]={0}, min=999, next_process=
    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("\n Given problem is not schedulable
        under the said scheduling algorithm\n");
    }
    for(int i=0; i<time; i++){
        min =1000;
        for(int j=0; j<num_of_process es; j++){
            if(remain_time[j] >0) {
                if(min > period[j]){
                    min = period[j];
                    next_process = j;
                }
```

```
    }
  }
if (remain_time[next_process]>0){
    process_list[i] = next_process+1;
    remain_time[next_process] -= 1;
}
for(int k=0; k<num_of_process; k++){
  if ((i+1) % period[k] == 0){
      remain_time[k] = execution_time[k];
      next_process = k;
  }
 }
}
 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[i])/
                    deadline[i];
  }
  int n= num_of_process;
  int process[num_of_process];
  int max_deadline, current_process=0,
  min_deadline, process_list[time];
  bool is_ready[num_of_process];
  for(int i=0; i<num_of_process; i++){
      is_ready[i]= true;
      process[i]= i+1;
  }
  max_deadine = deadline[0];
  for(int i=1; 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[i]<deadline[j]){
      int temp = execution_time[j];
      execution_time[j] = execution_time[i
      execution_time[i] = temp;
      temp = deadline[i];
      deadline[j] = deadline[i];
      deadline[i] = 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 t = 0; t <= time; t++){
  if (current_process != -1){
    --execution_time[current_process];
    process_lost[t] = process[current_process];
  }
  else
    process_lost[t] =0;
  for(int i=0; i< num_of_process; i++){
    --deadline[i];
    if ((execution_time[i] ==0)&& rs_ready[i]
      deadline[i] += remain_deadline[i];
```

```
        is_ready[i] = true;
    }
}
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("\n1. Rate Monotonic \n2. Earliest
        Deadline first \n3. Proportional scheduling
        \n\n Enter your choice: ");
        scanf("%d", &option);
        switch(option)
        {
        case 1: get_process_info(option);
                observation_time = get_observation_time
                                    (option);
                rate_monotonic(observation_time);
                break;
        case 2: get_process_info(option);
```

```
observation_time(option);
rate earliest_deadline_first(observation_time);
break;
case 3: exit(0);
default: printf("In Invalid statement")
}
}
return 0;
}
```

Output:

1. Rate monotonic
2. Earliest Deadline first

Enter your choice: 1

Enter total no. of process (max 10): 3

process 1:

Execution time: 1

period: 3

process 2:

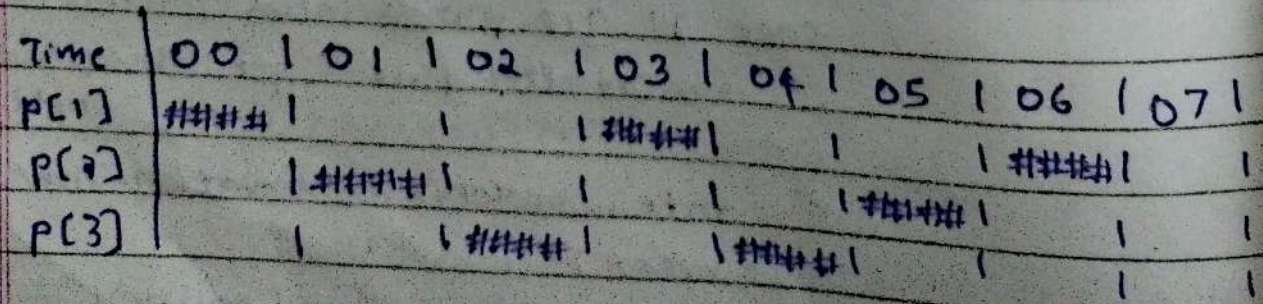
Execution time: 1

period 4

Process 3:

Execution time: 2

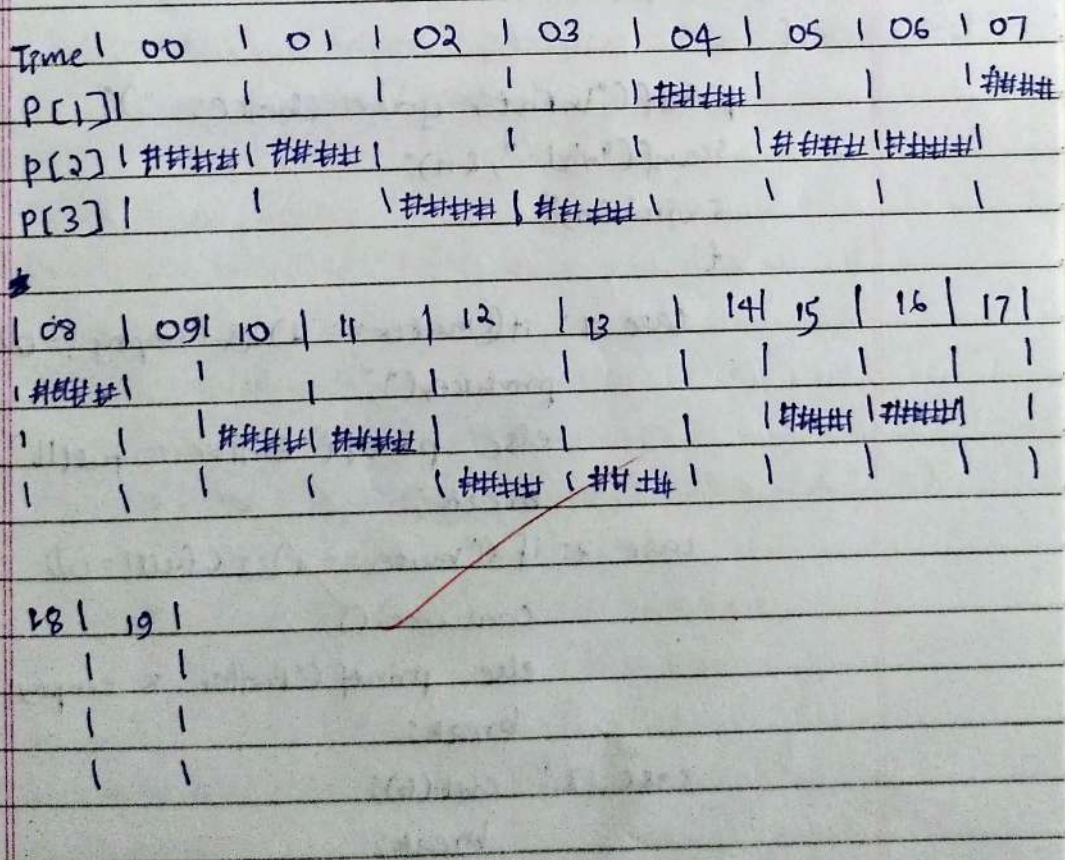
period: 8



Enter your choice: 2
Enter total no of processes (max 10): 3
process 1:
Execution time: 3
deadline: 20

process 2:
Execution time: 2
deadline: 5

Process 3:
Execution time: 2
deadline: 10



Enter your choice: 3

5. a) 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\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice: ");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                    producer();
                    else printf("Buffer is full!!");
                    break;
            case 2: if((mutex==1)&&(full!=0))
                    consumer();
                    else printf("Buffer is empty!!");
                    break;
            case 3: exit(0);
                    break;
        }
    }
    return 0;
```

```
}
int wait(int s)
{
  return (--s);
}
int signal(int s)
{
  return (++s);
}
void producer()
{
   mutex = wait(mutex);
   full = signal(full);
   x++;
   printf("\nProducer produces the item %d", x);
   mutex = signal(mutex);
}
void consumer()
{
  mutex = wait(mutex);
  full = wait(full);
  empty = signal(empty);
  printf("\nConsumer consumes item %d", x);
  x--;
  mutex = signal(mutex);
}
```

Output

1. Producer
2. Consumer
3. Exit

Enter your choice: 2
Buffer is empty!!

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

b) Write a C program to simulate the concept of dining philosophers problem.

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i+4)%N
#define RIGHT (i+1)%N
int state[N];
int phil[N]={0,1,2,3,4};
sem_t mutex;
sem_t S[N];
void test(int i)
{
    if(state[i]==HUNGRY && state[LEFT]!=EATING &&
```

```
state[RIGHT] != EATING) {
    state[i] = EATING
    sleep(2);
    printf("Philosopher %d takes fork %d and %d\n",
    i+1, LEFT+1, i+1);
    printf("Philosopher %d is Eating\n", i+1);
    sem_post(&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] = 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,
    &phil[i]);
    printf("Philosopher %d is thinking\n",i+1);
  }
  for(i=0;i<N;i++) pthread_join(thread_id[i],NULL);
}
```

Output:

```
Philosopher 1 is thinking
philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
"           1 is Hungry
"           2 is hungry
'           3 is hungry
"           4 is hungy
```

" 5 takes fork 4 and 5
" 5 is eating
" 5 putting fork 4 & 5 down
" 5 is thinking
" 4 takes fork 3 and 4
" 4 is eating
" 1 takes fork 5 and 1
" 1 is eating
" 4 putting fork 3 and 4 down
" 4 is thinking

12/6/24

Deadlock Avoidance

① Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.

```
#include<stdio.h>
int main(){
  int n,m,i,j,k;
  printf("Enter the number of processes: ");
  scanf("%d",&n);
  printf("Enter the number 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:\n");
  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(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 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 number of process : 5
Enter the number of resources : 3
Enter the allocation matrix:
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 3 2
Following is the safe sequence
P1 -> P3 -> P4 -> P0 -> P2

Deadlock Detection

Date: 13-6-24
Page: 33

(8) Write a C program to simulate deadlock detection.

```
#include<stdio.h>
static int marks[20];
int i, j, np, nr;
int main(){
  int alloc[10][10], request[10][10], avail[10], r[10], w[10];
  printf("\nEnter the no of processes: ");
  scanf("%d", &np);
  printf("\nEnter the no. of resources: ");
  scanf("%d", &nr);
  for(i=0; i<nr; i++){
     printf("\nTotal Amount of the Resource R%d: ", i+1);
     scanf("%d", &r[i]);
  }
  printf("\nEnter the request matrix:");
  for(i=0; i<np; i++){
    for(j=0; j<nr; j++){
       scanf("%d", &request[i][j]);
    }
  }
  printf("\nEnter 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];
    }
  }
```

```
//marking processes with zero allocation
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(j=0; j<np; 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 number of process: 5
Enter number of resources: 3
Total Amount of the Resources R1: 7
Total Amount of the Resource R2: 5
Total Amount of the Resource R3: 3
Enter the request matrix: 2 1 0
2 0 2
6 0 2
1 0 0
0 1 0
Enter the allocation matrix: 2 1 3
0 2 0
3 0 2
1 1 1
0 0 0
Deadlock detected

19/6/24

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

```
#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("Memory management scheme\n");
    printf("\nEnter the number of blocks:\n");
    for(int i=1; i<=nb; i++){
        scanf("%d", &b[i]);
    }
    printf("\nEnter the size of the files:\n");
    for(int i=1; i<=nf; i++){
        scanf("%d", &f[i]);
    }
    printf("\nMemory management scheme - First fit");
    firstFit(b, nb, f, nf);
    printf("\n\nMemory Management scheme - Worst Fit");
    worstFit(b, nb, f, nf);
    printf("\n\nMemory management scheme - Best fit");
    bestFit(b, nb, f, nf);
    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("\nFile_no:\tFile_size:\tBlock_no:\tBlock_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 worstfit(int b[], int nb, int f[], int nf){
  int bf[max]={0};
  int ff[max]={0};
  int frag[max], i, j, temp, highest=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){
          ff[i] = j;
          highest = temp;
        }
      }
    }
    frag[i] = highest;
```

```
        bf[ff[i]] = 1;
        highest = 0;
    }
    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_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 bestFit(int b[], int nb, int f[], int nf){
    int bf[max] = {0};
    int ff[max] = {0};
    int frag[max], i, j, temp, lowest = 10000;
    for(i=1; i<=nf; i++){
        for(j=1; j<=nb; j++){
            if(bf[j] != 1)
                temp = b[j] - f[i];
                if(temp >= 0 && lowest > temp){
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:
    \tFragment");
    for(i=1; i<=nf && ff[i]!=0; i++){
```

Date 3-7-24
Page 39

```
printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i],
b[ff[i]], frag[i]);
    }
}
```

Output:

Memory Management Schemes

Enter the number of blocks: 5

Enter the number of files: 8

Enter the size of the blocks

100 200 300 400 500

Enter the size of the files:

212 415 63 124 23 89 73 13

Memory management scheme - First Fit:

File no:	File Size:	Block no:	Block size:	Fragment
1	212	3	300	88
2	415	5	500	85
3	63	1	100	37
4	124	2	200	76
5	23	4	400	377
6	89	0	1969215917	2000000000
7	73	0	"	4096
8	13	0	"	4104

Memory management scheme - Worst Fit:

File no:	File Size:	Block no:	Block size:	Fragment
1	212	5	500	288
2	415	0	1969215917	0
3	63	4	400	337
4	124	3	300	176
5	23	2	200	177
6	89	1	100	11

7	13	0	1969258917	0
8	13	0	"	0

Memory Management Scheme - Best Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	212	3	300	88
2	415	5	500	85
3	63	1	100	37
4	124	2	200	76
5	93	4	400	307

2 Write a C program to simulate page replacement algorithms:

(a) FIFO

(b) LRU

(c) Optimal

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

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

void initialize(){
    pgfaultcnt=0;
    for(i=0; i<f; i++)
        p[i]=9999;
}

int isHit(int data){
    hit=0;
    for(j=0; j<f; j++){
```

```
        if(p[j]==data){
            hit=1;
            break;
        }
    }
    return hit;
}

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

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

void dispPgFaultCnt(){
    printf("\nTotal no of page faults: %d", pgfaultcnt);
}

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++)
            p[k]=p[k+1];
        p[k]=in[i];
        pgfault cnt++;
        displages();
    }
    else printf("No page fault");
}
displgFault Cnt();
}

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<f; 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<n;j++){
        if(near[j]>max){
            max=near[j]; repindex=j; }}
    p[repindex]=in[i];
    pgfaultcnt++;
    dispPages(); } else printf("No page fault!!"); }
    dispPgFaultCnt(); }
void lru(){
    initialize(); int least[50];
    for(i=0;i<n;i++){
        printf("\nfor %d ",in[i]);
        if(search(in[i])==0){
            for(j=0;j<nf;j++){
                int pg=p[j];
                int found=0;
                for(k=f-1;k>=0;k--){
                    if(pg==rf[k]){
                        least[j]=k;
                        found=1; break; }
                    else found=0; }
                if(!found) least[j]=-9999; }
            int repindex;
            for(j=0;j<nf;j++){
                if(least[j]<min){
                    min=least[j];
                    repindex=j;
                }}
            p[repindex]=in[i];
            pgfaultcnt++;
            dispPages(); }
            else printf("No page fault!");
    }
```

```
dispPgFaultCnt(); }
int main(){
    int choice;
    while(1)
    {
    printf("\nPage Replacement Algorithms\n1. Enter data\n2.
    Fifo\t3. Optimal\t4. LRU\t5. Exit\nEnter your choice:");
    scanf("%d", &choice);
    while(1){
        scanf("%d", &choice);
        switch(choice){
            case 1: getData();
                    break;
            case 2: Fifo();
                    break;
            case 3: optimal();
                    break;
            case 4: exit();
            default: return 0;
            break;
        }
    }
}
```

Output: ~~Enter no. of frames~~ Page Replacement Algorithms

1. Enter data
2. FIFO
3. Optimal
4. LRU
5. Exit

Enter your choice: 2

Enter length of page reference sequence : 12

Enter sequence 1 2 3 4 1 2 5 1 2 3 4 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: 253

IMPORTANT NOTES

For q: 5 3 4

For 5: No page fault

~~For 6:~~ Total no. of page fault: 9

~~Page Replacement Algorithms~~

~~1. Enter data~~

~~2. FIFO~~

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: 235

For 4: 425

For 5: ~~345~~ No page fault

Total no. of page faults: ~~8~~ 7

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

5

3/7/24

FCFS Disk scheduling

classmate
Date 10/7/24
Page 47

a)
```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int RQ[100], i, n, TotalHeadMoment = 0, initial;
    printf("Enter the number of requests \n");
    scanf("%d", &n);
    printf("Enter the Request Sequence \n");
    for(i = 0; i < n; i++)
    scanf("%d", &RQ[i]);
    printf("Enter the 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 moment is %d", TotalHeadMoment);
    return 0;
}
```

Output: Enter the number of Requests 5
Enter the request sequence: 34 60 120 180 240
Enter the initial head position 50
Total head moment is 222

SCAN Disk Scheduling

b)

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int RQ[100], i, j, n, TotalHeadMoment=0, initial, size,
    move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Request Sequence\n");
    for(i=0; i<n; i++)
        scanf("%d",&RQ[i]);
    printf("Enter the initial head position\n");
    scanf("%d",&initial);
    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;
            }
        }
    }
```

```
int index;
for(i=0; i<n; i++){
  if(initial<RQ[i]){
    index = i;
    break;
  }
}

if(move==1){
  for(i= index; i<n; i++){
    TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
    initial = RQ[i];
  }
  TotalHeadMoment = TotalHeadMoment + abs(size - RQ[i-1] - 1);
  initial = size - 1;
  for( i = index - 1; i >= 0; 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);
  initial = 0;
  for(i=index; i<n; i++) {
      TotalHeadMoment = TotalHeadMoment + abs(RQ[i]
      -initial);
      initial = RQ[i];
    }
  }
  printf("Total Head movement is %d", TotalHeadMoment);
  return 0;
}
```

Output: Enter the number of request: 5
Enter the request sequence: 34 60 120 180 240
Enter the initial head position 50
Enter the total disk size
250
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 414.

C-SCAN

classmate
Date 10/7/24
Page 51

c)
```
#Include<studio.h>
#Include<stdlib.h>
int main(){
    Int RQ[100], i,j,n, TotalHeadMoment=0, initialize, size, move;
    printf("Enter the number 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 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-j-1; j++){
       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 (initial < RQ[i]) {
        index = i;
        break;
    }
}
if (move == 1) {
    for (i=index; i<n; i++) {
        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 movement is %d", TotalHead
                                    -Moment);
return 0;
}
```

Output: Enter the number of requests 5
Enter the sequence of requests 34 60 120 180 240
Enter the initial head position: 50
Enter the total disk size 250
Enter the head movement direction: 1
Total head movement is 482.

10/7/24