INDEX

Name [redacted] AGAM TIWARI Sub. OS LAB

Std.: 4th Sem Div. A Roll No. CS023

Telephone No. E-mail ID.

Blood Group. B- Birth Day.

Sr.No.	Title	~~Page No.~~ Date	Sign./Remarks
1	Matrix Calculations	8/5/24	10
2	CPU Scheduling 1) FCFS 2) SJF	15/5/24	
3	Priority / Round Robin	22/5/24	8
4	Rate Monotonic / Earliest - D First	05/6/24	
5	Producer - Consumer Problem	12/6/24	10
6	Dining - Philosophers	12/6/24	
7	Banker's Algorithm for Deadlock Avoidance	19/6/24	8
8	Deadlock detection	19/6/24	9
9	Contiguous Memory Allocation	3/7/24	
10	Page Replacement Algorithm	10/7/24	

PAGE NO :
DATE :

8/5/24

Q Addition, Subtraction and Multiplication of Matrices.

⇒
```
#include <stdio.h>
#define ROWS 3
#define COLS 3

void printMatrix(int matrix[][COLS]) {
        for (int i=0; i<ROWS; i++) {
            for (int j=0; j<COLS; j++) {
                printf("%d \t", matrix[i][j]);
            }
            printf("\n");
        }
}

void addMatrices(int matrix1[][COLS], int matrix2[][COLS],
        int result matrix[][COLS]) {

        for(int i=0; i<ROWS; i++) {
            for (int j=0; j<COLS; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }
}

void SubtractMatrices (int matrix1[][COLS], int matrix2,
                int result[][COLS]) {
        for (int i=0; i<ROWS; i++) {
            for (int j=0; j<COLS; j++) {
                result[i][j] = matrix1[i][j] - matrix2[i][j];
            }
        }
}
```

PAGE NO:
DATE:

```
void multiplyMatrices (int matrix1[][COLS],
                       int matrix2[][COLS],
                       int result[][COLS]){
    for (int i=0; i<ROWS; i++){
        for (int j=0; j<COLS; j++){
            result[i][j]=0;
            for (int k=0; k<COLS; k++){
                result[i][j] += matrix1[i][k]
                                * matrix2[k][j];
}   }   }   }

int main(){
    int matrix1[ROWS][COLS];
    int matrix2[ROWS][COLS];
    int result[ROWS][COLS];

    printf("~~int~~ Enter elements of matrix 1:\n");

    for (int i=0; i<ROWS; i++){
        for (int j=0; j<COLS; j++){
            printf("matrix1[%d][%d]:", i,j);
            scanf("%d", &matrix1[i][j]);
        }
    }

    addMatrices(matrix1, matrix2, result);
    printf("Add Result:\n");
    printMatrix(result);

    subtractMatrices(matrix1, matrix2, result);
    printf("Subtract Result:\n");
    printMatrix(result);
```

PAGE NO :
DATE :

```
multiplyMatrices(matrix1, matrix2, result);
printf("Multiplication Result:\n");
printMatrix(result);
}
```

8/5/24

Output :

PAGE NO :
DATE :

15/5/24

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

Code ⇒

```
#include <stdio.h>
int n,i,j,pos,temp,choice, Burst_time[20], Arrival_Time[20],
Waiting_time[20], Turn_around_time[20]; process[20],
total = 0;
float avg_Turn_around_time = 0, avg_Waiting_time = 0;
avg_Arrival_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 Arrival Time \t\t Turnaround Time");
    for(i=0; i<n; i++) {
        Turn_around_time[i] = Burst_time[i]+waiting_time[i];
        avg_Waiting_time += Turn_around_time[i];
        printf("\n P[%d]\t\t %d \t\t\t %d",i+1,
            Burst_time[i], Waiting_time[i],
            Turn_around_time[i]);
    } avg_Arrival_time = (float)(avg_Arrival_time)/(float)i;
    avg_Waiting_time = (float)(avg_waiting_time)/(float)i;
    avg_Turn_around_time = (float)(avg_Turn_around_time)
                /(float)i;
    printf("\n Avg Waiting Time : %.2f", avg_Waiting_time);
    printf("\n Avg Turnaround Time %.2f \n",
            avg_Turn_around_time);
}
```

PAGE NO :
DATE :

```
int SJF() {
    for(i=0; i<n; i++) {
        pos = i
        for(j=i+1; j<n; i++) {
            if(Burst_time[j] < Burst_time[pos])
                pos = j ;
        }
        temp = Burst_time[i];
        Burst_time[i] = Burst_time[pos];
        Burst_time[pos] = temp;
        temp = process[i];   temp = Arrival_time[i]
        process[i] = process[pos]; temp
        process[pos] = temp;  Arrival_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\t %d \t\t\t %d",
            process[i], Burst_time[i], Waiting_time[i],
            Turn_around_time[i]); Arrival_time
                                [i]);
    }
```

PAGE NO :
DATE :

```
    avg_Turn_around_time = (float) total/n;
    printf("\n\n Avg Waiting Time = %f", avg_Waiting_time);
    printf("\n Avg Turnaround Time = %f", avg_Turn_time);
}

int main() {
    printf("Enter the total number of processes :");
    scanf("%d", &n);
    printf("\n Enter Burst Time and Arrival Time:")
    for(i=0; i<n; i++) {
        printf("P[%d]:", i+1);
        scanf("%d %d", &Burst_time[i], &Arrival_time[i]);
        process[i] = i+1;
    }
    while(1) {
        printf("\n -- MAIN MENU -- \n");
        printf("1. FCFS Scheduling \n 2. SJF Scheduling\n");
        printf("\n Enter your choice :");
        scanf("%d", &choice);
        switch (choice) {
            case 1: FCFS(); break;
            case 2: SJF(); break;

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

PAGE NO :
DATE :

Output :

Enter the total no. of processes : 4

Enter Arrival Time and Burst Time :
P[1] Arrival Time : 0
P[1] Burst Time : 3
P[2] Arrival Time : 1
P[2] Burst Time : 6
P[3] Arrival Time : 4
P[3] Burst Time : 4
P[4] Arrival Time : 6
P[4] Burst Time : 2

-- MAIN MENU --
1. FCFS Scheduling
2. SJF Scheduling

Enter choice : 1

Process	Arrival Time	Burst Time	Waiting Time	Turn Time
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	5	9
P[4]	6	2	7	9

Avg Waiting Time : 3.50
Avg Turaround Time : 7.25

PAGE NO :
DATE :

-- MAIN MENU --

1. FCFS Scheduling
2. SJF Scheduling

Enter your choice : 2

Process	Arrival Time	Burst Time	Waiting Time	Turn Time
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	7	11
P[4]	6	2	3	5

Avg Waiting Time = 3.00
Avg Turnaround Time = 6.75

15/5/24

22/5/24

PAGE NO :
DATE :

f CPU Scheduling

a) 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 number of processes :");
    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 ("Enter the priority :");
        scanf ("%d", &proc[i].priority);
    }
```

PAGE NO :
DATE :

```
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 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;
    }
}
```

PAGE NO :
DATE :

b) Round Robin

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

int turnaroundtime (int processes [], int n ....) {
    for (int i=0 ; i<n ; i++)
        tat[i] = bt[i] + wt[i] ;
        return 1;
}
int waiting time (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] -= quantum ;
                }
```

PAGE NO :
DATE :

```
        if (done == true)
            break;
        }
        return 1;
}

int findavgTime (int processes [], int n, int bt [],
{   int wt[n], tat [n], total_wt = 0,
    total_tat = 0;

    waiting time (processes, n, bt, wt, quantum);
    turnaround time (processes, n, bt, wt, at);

    printf ("\n\n Processes \t \t Burst time ");
    for (int i=0; i<n; i++){
        total_wt = total_wt + wt [i]
        total_tat = total_tat + bt (i);
        printf ("\n \t %d %d %d %d,
                i+1, bt [i], wt [i], tat [i]);
    }

    printf (" Processes Burst Time \t
            Waiting Time turnaround
            time \n");

}
```

PAGE NO :
DATE :

```
int main() {
    int n, processes[n], burst_time[n], quantum;
    printf("Enter the Number of Processes :");
    scanf("%d", &n);

    printf("Enter the quantum time :");
    scanf("%d", &quantum);

    int i=0;
    for(i=0; i<n; i++){
        printf("\n Enter 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 : (Round Robin)

Enter the no. of process : 3

Enter quantum : 2

P	BT
1	5
2	7
3	3

PID	BT	WT	TAT
1	5	7	12
2	7	8	15
3	3	8	11

PAGE NO :
DATE :

Avg WT : 7.667
Avg TAT : 12.667

Output (Priority)

Enter the no. of processes : 4
Enter the arr. time for process 1 : 0
" " burst time for process 1 : 4
" " priority " " 1 : 2

" " Arrival " " 2 : 1
" " burst " " " : 3
priority 3

" " Arrival " " 3 : 2
" " Burst " " 3 : 1
" " priority " " 3 : 4

" " Arrival " " 4 : 3
" " Burst " " 4 : 5
" " priority " " 4 : 5

Process	Burst time	Arrival time	Priority	WT	TT
1	4	0	2	7	11
2	3	1	3	9	12
3	1	2	4	4	5
4	5	3	5	10	15

Avg Waiting Time = 7.4
Avg Turnaround time = 10.4

PAGE NO :
DATE :

5/6/24

Q Write a C Program for CPU Scheduling Algorithms :

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 d; int burst_time; float priority;
} Task;

int no_of_process, 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 processes (max %d):",
            MAX_PROCESS);
    scanf("%d", &num_of_process);
    if (num_of_process<=0)
        exit(0)
    for(int i=0; i<num_of_process; i++) {
        printf("\n Process %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]);
        }}
}
```

PAGE NO :
DATE :

```
~~int max~~
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;
}

void rate_monotonic (int time) {
    int process_list[100] = {0}, min = 999;
    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 ("In 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 ; j++) {
            if (remain_time[j] >0) {
                if (min > period[j]) {
                    min = period[j];
                    next_process = j;
            }   }
        }
```

PAGE NO :
DATE :

```
        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_deadline = deadline [0] ;
    for (int i=1 ; i < num_of_process ; i++) {
        if (deadline [i] > max_deadline)
            max_deadline = deadline [i] ;
    }
```

PAGE NO :
DATE :

```
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] = 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_list [t] = process [current_process] ;
}
    else
        process_list [t] = 0 ;

    for (int i=0 ; i< num_of_process ; i++){
        --deadline [i] ;
        if ((execution_time [i] == 0) &&
            is_ready [i]) {
            execution_time [i] = remain_time [i] ;
            is_ready [i] = true ;
} }
```

PAGE NO :
DATE :

```
        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 \n 3. 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 = get_observation_
                    time(option);
                    earliest_deadline_first(observation_
                    time);
                    break;
            case 3: exit(0);
        } }
```

PAGE NO :
DATE :

Output :

1. Rate Monotonic
2. Earliest Deadline First

Enter your choice : 2
Enter total no. of process (max 10) : 3

Process 1 :
Execution time : 3
deadline : 20

Process 2 : Execution time : 2 deadline : 10

Time	00	01	02	03	04	05	06	07	08	09
p[1]					###			###	###	
p[2]	###	###				###	###			
p[3]			###	###						

Enter choice : 1
Enter total no. of process (max 10) : 3
Process 1 :
Execution Time : 1
Period : 3

Process 2 :
Execution time : 1
period : 4

12/6/24

Process 3 :
Execution time : 2
period : 8

Time	00	01	02	03	04	05	06
P[1]	###			###			###
P[2]		###				###	
P[3]			###		###		

12/6/24

PAGE NO :
DATE :

Q Write a C program to stimulate the concept of 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("\n 1. 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 : exit(0);
                        break;
    } }
}
```

PAGE NO :
DATE :

```
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 ("In Producer produces the item %d", x);
    mutex = signal (mutex);
}
void consumer () {
    mutex = wait (mutex);
    full = wait (full);
    empty = signal (empty);

    printf ("In Consumer consumes item %d", x);
    x--;
    mutex = signal (mutex);
}
```

<u>Output</u> : 1. Producer 2. Consumer 3. Exit
Enter choice : 1
Producer produces item 1
Enter choice : 1
Producer produces item 2
Enter choice : 1
Producer Produces item 3
Enter choice 1
Buffer is full

PAGE NO :
DATE :

Enter choice : 2

Customer customer, item 3

12/6/24

PAGE NO :
DATE :

12/6/24

Q 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]);
    }
}
```

PAGE NO :
DATE :

```
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 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);
        putfork (*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);
```

PAGE NO :
DATE :

```
for(i=0; i<N; i++) {
    pthread_create(&thread_id[i], NULL,
    philosopher, &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
  P2        "   "    "
  P3        "   "    "
  P4        "   "    "
  P5 is Hungry
  P1        "   "
  P2        "   "
  P3        "   "
  P4 taken fork 3 and 4
  P4 is eating
  P4 is putting fork 3 x 4 now
  P4 is thinking
  P3 takes fork 2 x 3
  P3 is eating
```

PAGE NO :
DATE :

19/6/24

Q 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;j++)
        f(k)=0
    int need[n][m];
```

PAGE NO :
DATE :

```
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;
} }
```

PAGE NO :
DATE :

```
    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;
}
```

<u>Output:</u>

Enter the no. of processes : 5
Enter the no. 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 3
11 3 3

PAGE NO :
DATE :

Enter the available resources :
3 3 2

Following is the SAFE Sequence
$P1 \rightarrow P2 \rightarrow P4 \rightarrow P0 \rightarrow P3$

PAGE NO :
DATE :

19/6/24

Q Write a C program to simulate deadlock:

```
⇒ #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("Enter no. of process :")
    scanf("%d", &np);
    printf("Enter no. of resources :");
    scanf("%d, &nr);
    for(i=0; i<nr; i++) {
        printf("\n Total Amount of the resource R %d :", i+1)
        scanf("%d", &r[i]);
    }
    printf("\n Enter the resource 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(i=0; j<np; i++) {
            avail[j] -= alloc[i][j];
    } }
```

PAGE NO :
DATE :

```
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 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 ;
```

PAGE NO :
DATE :

```
    if(deadlock)
        printf("\n Deadlock detected");
    else
        printf("\n Deadlock possible");
}
```

Output:

Enter the no. of processes : 5
Enter the no. 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 request matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter the allocation matrix
0 0 1
1 0 0
1 3 5
0 6 3
0 0 1

Deadlock detected.

3/7/24

PAGE NO :
DATE :

2/7/24

Q. Contiguous Memory Allocation
a) Worse fit
b) Best fit
c) First fit

```
#include <stdio.h>
# define max 25
void firstFit (int b[], int nb, int f[], int nf);
void worseFit (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 Managment Schemes \n"),
    printf ("\n Enter the number of blocks :");
    scanf ("%d", &nb);

    printf ("\n Enter the number of files :");
    scanf ("%d", &nf);
    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++) {
        scanf ("%d", &f[i]);
    }
    printf ("\n Memory Managment Scheme -
                First Fit");
    firstFit (b, nb, f, nf) ;
```