INDEX

Name : KSHITIZ ANAND Class : 4C

Section : Roll No. : Subject : OS

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
01	8/05/24	FCFS scheduling.	01	
02	15/05/24	STF	02	
03	15/05/24	Priority scheduling	04	
04	15/05/24	Round Robin	06	
05	22/05/24	Multilevel queue	08	
06	29/05/24	Rate Monotonic	09	
		Earliest deadline first	11	
		Proportional scheduling	13	
07	12/06/24	Producer - Consumer problem	15	
08	19/06/24	Dining - Philosopher problem	16	
		Bankers algorithm.	18	
08	3/07/24	Deadlock detection	19	
09	3/07/24	Continuous memory allocation	21	
10	10/07/24	Page Replacement algorithm	23	

10/7 completed

LAB 1

Q1 a) FCFS scheduling

```
# include <stdio.h>

void CompTime (int n, int a[], int b[], int c[])
{
    int t=0;
    for (int i=0; i<n; i++) {
        if (a[i] > t){
            t = a[i];
            t += b[i];
            c[i] = t;
        }
    }
}
void tatTime (int n, int a[], int c[], int t[])
{
    for (int i=0 ; i<n; i++)
        t[i] = c[i] - a[i];
}
void waitTime (int n, int b[], int t[], int w[])
{
    for (int i=0 ; i<n; i++)
        w[i] = t[i] - b[i];
}
void avgTime (int n, int a[], int b[])
{
    int c[n], t[n], w[n];
    int sumTAT = 0, sumWT=0;
    compTime (n,a,b,c);
    tatTime (n,a,c,t);
```

```
waitTime(n, b, t, w);
printf("P A B C T W \n");
for (int i=0; i<n; i++){
    printf("%d %d %d %d %d %d \n",
            i+1, a[i], b[i], c[i], t[i], w[i]);
    sumTAT+=t[i];
    sumWT+=w[i];
}
float avgTAT = (float) sumTAT/n;
float avgWT = (float) sumWT/n;
printf("\n Average TAT: %f\n", avgTAT);
printf("Average WT: %f \n", avgWT);
}

int main(){
    int n=4
    int a[]={0,1,5,6};
    int b[]={2,2,3,4};

    avgTime(n,a,b);
    return 0;
}
```

op

P	A	B	C	T	W
1	0	2	2	2	0
2	1	2	4	3	1
3	5	3	8	3	0
4	6	4	12	6	2

Average TAT : 3.500000
Average WT : 0.750000

Q16 SJF (non-preemptive)

```
#include <stdio.h>

void findCompletionTime (int processes[], int n, int bt[],
        int at[], int wt[], int tat[], int rt[], int ct[])
{
  int completion[n];
  int remaining[n];

  for (int i=0 ; i<n ; i++)
      remaining[i] = bt[i];

  int currentTime = 0;
  for (int i=0; i<n ; i++)
  {
    int shortest = -1;
    for (int j=0 ; j<n ; j++)
    {
      if ( at[j]<= currentTime && remaining[j] >0)
      {
        if (shortest == -1 || remaining[j] <
                        remaining[shortest])
          shortest = j;
      }
    }
    if (shortest == -1)
    {
      currentTime++;
      continue ;
    }
```

P.T.O

```
completion [shortest] = current Time +
                        remaining [shortest];
currentTime = completion [shortest];
wt [shortest] = currentTime - bt [shortest]
                - at [shortest];
tat [shortest] = current Time - at [shortest];
rt [shortest] = wt [shorter];
remaining [shortest] = 0;
}
printf("Process \t Arrival \t Burst \t Waiting
        \t Turnaround \t Response \t Completion);
for (int i=0; i<n; i++)
{
  ct[i] = completion [i];
  printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \n",
      process[i], at[i], bt[i], wt[i], tat[i],
      rt[i], ct[i]);
}
}

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

int process [n];
int burst-time [n];
int arrival-time [n];
printf("Enter process Number: \n");
for (int i=0; i<n; i++)
        scanf("%d", & process [i]);
```

P.T.O

```
printf ("Enter Arrival time : \n");
for (int i=0 ; i<n ; i++)
    scanf ("%d", &Arrival_time[i]);
printf ("Enter Burst Time : \n");
for (int i=0 ; i<n ; i++)
{
    scanf ("%d", &burst_time[i]);

int wt[n], tat[n], rt[n], ct[n];
for (int i=0 ; i<n ; i++)
    rt[i]=-1 ;
printf ("\n STF (Non-Preemptive scheduling : \n");
findCompletionTime (process, n, burst_time, arrival_
        time, wt, tat, rt, ct);
}
```

Op

Enter number of process : 5
Enter Process Number :
1 2 3 4 5
Enter Arrival time :
2 1 4 0 2
Enter Burst time :
1 5 1 6 3
STF (Non Preemptive scheduling)

Process	Arrival	Burst	Waiting	Turnaround	Response	Comp
1	2	1	4	5	4	7
2	1	5	10	15	10	16
3	4	1	3	4	3	8
4	0	6	0	6	0	6
5	2	3	6	9	6	11

Average turnaround time : 7.800000
Average waiting time : 4.600000

Q a Priority

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

void findCompletionTime(int process[], int n, int bt[]
                        int at[], int wt[], int tat[], int ct[],
                        int rt[], int priority[], bool
                                    isLowerPriorityHigher)
{
  int remaining[n];
  int currentTime = 0;
  int completed = 0;
  bool isFinished[n];

  for(int i=0 ; i<n; i++){
      remaining[n]= bt[i];
      isFinished[i]= false;
      rt[i] =-1;
  }
  while (completed<n){
      int highestPriorityIndex =-1;
      int highestPriority = isLowerPriorityHigher ?100000:-1;

      for (int i=0; i<n ;i++){
          if (at[i] <= currentTime && !isFinished[i]&&
           (!isLowerPriorityHigher && priority[i]<highestPriority)
          ||(!isLowerPriorityHigher && priority[i] > highestPriority))
          {
            highestPriority = priority[i];
            highestPriorityIndex = i;
          }
  }
```

```
if (highestPriorityIndex == -1) {
    currentTime++;
    continue;
}
int currentProcess = highestPriorityIndex;
if (rt[currentProcess] == -1) {
    rt[currentProcess] = currentTime - at[currentProcess];
}
remaining[currentProcess]--;
currentTime++;
if (remaining[currentProcess] == 0) {
    isFinished[currentProcess] = true;
    completed++;
    ct[currentProcess] = currentTime;
    tat[currentProcess] = ct[currentProcess] -
                          at[currentProcess];
    wt[currentProcess] = tat[currentProcess] -
                          bt[currentProcess];
}
}
for (int i=0; i<n; i++) {
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
        process[i], at[i], bt[i], priority[i], ct[i], tat[i], wt[i],
        rt[i]);
}
}
void main()
{
    int n;
    printf("Enter the number of processes : ");
    scanf("%d", &n);
```

PTO

```
int processes[n];
int burst_time[n]
int arrival_time[n];
int priority[n];
int priorityType;
bool islowerPriorityHigher;
printf("Enter 1 if lower number indicates higher
  priority: ");
scanf("%d", &PriorityType);
islowerPriorityHigher = (priorityType == 1);
printf("Enter Process Number : \n");
for (int i=0; i<n; i++){
    scanf("%d", &processes[i]);
}

printf("Enter Priority : \n");
for(int i=0; i<n; i++) {
   scanf("%d", &priority[i]);
}

printf("Enter Burst Time : \n");
for (int i=0; i<n; i++) {
  scanf("%d", &burst_time[i]);
}

int wt[n], tat[n], ct[n], rt[n];
printf("\n Priority Scheduling (Preemptive): \n");
findCompletionTime(process, n, burst_time,
arrival_time, wt, tat, ct, rt, priority,
islowerPriorityHigher);
}
```

Output:

Enter the number of processes : 4
Enter 1 if lower number indicates higher priority,
0 if higher number indicates higher priority : 0
Enter process number : 1 2 3 4
Enter priority :
10 20 30 40
Enter Arrival Time :
0 1 2 4
Enter Burst Time :
5 4 2 1

Priority scheduling (Preemptive):

Process	AT	BT	Priority	CT	TAT	WT	RT
1	0	5	10	12	12	7	0
2	1	4	20	8	7	3	0
3	2	2	30	4	2	0	0
4	4	1	40	5	1	0	0

Q2b Round Robin

```
#include <stdio.h>
#include <stdbool.h>
void findCompletionTime (int process[], int n,
 int bt[], int at[], int wt[], int tat[],
 int ct[], int rt[], int quantum)
{
  int remaining [n];
  bool firstResponse [n];
  int currentTime =0;
  int completed =0;
  for (int i=0; i<n; i++){
    remaining[i] = bt[i];
    firstResponse [i] = true;
}
  int queue[n];
  int front =-1, rear =-1;
  void enque (int process) {
    if (rear == n-1)
        rear = -1;
    queue[++rear] = process;
    if (front == -1)
        front=0;
}
  int deque() {
    int process = queue[front];
    if (front == rear)
        front = rear = -1;
    else {
      front++;
      if (front == n)
        front=0;
}
```

```
return process;
}
bool inQueue[n];
for (int i=0; i<n; i++)
    inQueue[i] = false;
while (completed <n) {
    for (int i=0; i<n; i++) {
        if (at[i] <= currentTime && !inQueue[i]) {
            enqueue(i);
            inQueue[i] = true;
        }
    }
if (front == -1) {
    currentTime ++;
    continue;
}
int currentProcess = dequeue();
if (firstResponse[currentProcess]) {
    rt[currentProcess] = currentTime - at[current
                                        Process];
    firstResponse[currentProcess] = false;
}
if (remaining[currentProcess] > quantum) {
    remaining[currentProcess] -= quantum;
    currentTime += quantum;
}
else {
    currentTime += remaining[currentProcess];
    remaining[currentProcess] = 0;
    completed ++;
    ct[currentProcess] = currentTime;
    tat[currentProcess] = ct[currentProcess] -
                                at[currentProcess];
```

```
~~tat [currentProcess] = at~~
    wt [currentProcess] = tat [current Process] -
                                bt [currentProcess];
    for (int i=0 ; i<n ; i++) {
        if ( at [i] <= currentTime && ! inQueue[i])
        {
            enqueue[i];
            inQueue [i] = True;
        }
    }
    if (remaining [currentProcess] >0) {
        enqueue (currentProcess);
    }
}
printf (" Process \t Arrival Time \t Burst Time \t
        Completion Time \t TurnAroundTime \t
        WaitingTime \t Response Time \n");
for (int i=0 ; i<n ; i++) {
    printf(" %d \t\t %d \t\t %d \t\t %d
        \t\t %d \n", process[i], at [i], bt[i],
        ct[i], tat[i], wt[i], rt[i]);
    }
}
void main() {
    int n;
    printf(" Enter the number of process: ");
    scanf(" %d", &n);
    int process [n];
    int burst_time[n];
    int arrival_time [n];
    printf(" Enter process Number : \n");
```

```
for (int i=0; i<n; i++){
    scanf ("%d", &processes[i]);
}
printf ("Enter Arrival Time : \n");
for (int i=0; i<n; i++){
    scanf ("%d", &arrival_time[i]);
}
printf ("Enter Burst Time : \n");
for (int i=0; i<n; i++){
    scanf ("%d", &burst_time[i]);
}
int quantum;
printf ("Enter the time quantum : ");
scanf ("%d", &quantum);
int wt[n], tat[n], ct[n], rt[n];
printf ("\n Round Robin Scheduling : \n");
find CompletionTime (process, n, burst_time,
    arrival_time, wt, tat, ct, rt, quantum);
}
```

O/P

```
Enter the number of processes : 5
Enter process Number : 1 2 3 4 5
Enter Arrival Time : 0 1 2 3 4
Enter Burst Time : 5 3 1 2 3
Enter the time quantum : 2
Round Robin Scheduling :
```

Process	AT	BT	CT	TAT	WT	RT
1	0	5	13	13	8	0
2	1	3	12	11	8	1
3	2	1	5	3	2	2
4	3	2	9	6	4	4
5	4	3	14	10	7	5

Average Turn Around Time : 8.600000
Average Waiting Time : 5.800000

LAB-3 22/05/24

↳ Multilevel queue scheduling algorithm.

```
#include <stdio.h>

void sort ( int proc_id [], int at [], int bt[],
                int n ){
{
    int temp = 0;
    for (int i=0; i<n; i++){
        for (int j=i; j<n; j++)
        {
            if( at[j] < at [i] )
            {
                temp = at[i]; at[i] = at[j];
                at[j] = temp ;
                temp bt[j]; bt[j] = bt[i]; bt[i] = temp;
                temp = proc_id[i] ; proc_id[i]= proc_id[j];
                proc_id[j] = temp ;
            }
        }
    }
}


void fcfs ( int at [], int bt[], int ct[], int tat[],
        int wt[], int n, int *c )
{
    double ttat=0.0, twt=0.0;
    for (int i=0; i<n; i++){
        if (*c >= at[i])
            *c += bt[i];
        else
            *c = at[i] - ct[i-1] + bt[i];
        ct[i] = *c;
}
```

```
for (int i=0; i<n; i++)
    tat[i] = ct[i] - at[i];
for (int i=0; i<n; i++)
    wt[i] = tat[i] - bt[i];
}

void main(){
    int sn, un, c=0, int n=0;
    printf("Enter number of system process: ");
    scanf("%d", &sn); n=sn;
    int sproc_id[n], sat[n], sbt[n], sct[n], stat[n],
    swt[n];
    for (int i=0; i<sn; i++)
        sproc_id[i] = i+1;
    printf("Enter arrival times of the system
        process: \n");
    for (int i=0; i<sn; i++)
        scanf("%d", &sat[i]);
    printf("Enter burst time of system process: \n");
    for (int i=0; i<sn; i++)
        scanf("%d", &sbt[i]);
    printf("Enter number of user process: \n");
    scanf("%d", &un); n=un;
    int uproc_id[n], uat[n], ubt[n], uct[n], utat[n]
        uwt[n];
    for (int i=0; i<un; i++)
        scanf("%d", &ubt[i]);

    sort (sproc_id, sat, sbt, sn);
    sort (uproc_id, uat, ubt, un);

    fcfs (sat, sbt, sct, stat, swt, sn, &c);
    fcfs (uat, ubt, uct, utat, uwt, un, &c);
```

```
printf("\n scheduling :\n");
printf("system processes : \n");
printf("PID \tAT \t BT \t CT \t TAT \tWT
                                \n");

for (int i = 0; i<sn; i++)
    printf("%d\t %d \t %d \t %d \t %d \t %d \t
        sproc_id[i], sat[i], sbt[i], sct[i], stat[i], swt[i]);
printf("User processes : \n");
for (int i=0; i<un; i++)
    printf("%d \t %d \t %d \t %d \t \n",
    uproc_id [i], uat [i], ubt [i], uct[i], utat[i],
                uwt[i]);

}
```

o/p

Enter number of system process : 2
Enter arrival time of system process :
0 2
enter Burst times of system process :
4 3
Enter number of processes : 3
Enter arrival times of the user processes :
1 3 5
Enter burst times of user process :
2 3 1

Scheduling :
System process :

PID	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	2	3	7	5	2

User process :

PID	AT	BT	CT	TAT	WT
1	1	2	9	8	6
2	3	3	12	9	6
3	5	1	13	8	7

Date
Page 29/05/24
9

LAB - 4

4 1) Rate Monotonic

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

void sort (int proc [], int b[], int pt[], int n)
{
    for (int i=0 ; i<n ; i++ ){
        for (int j=i ; j<n ; j++ ){
            if (pt[j] < pt[i]) {
                int temp = proc [i];
                proc[i] = proc [j];
                proc [j] = temp;

                temp = b [i];
                b [i] = b [j]
                b[j] = temp;

                temp = pt [i],
                pt [i] = pt [j]
                pt [j] = temp;
            }
        }
    }
}

int lcmul(int p[], int n) {
    int lcm = p[0];
    for (int i=1 ; i<n; i++) {
        int a = lcm , b = p[i];
        while (b != 0) {
            int r = a % b;
            a = b;
            b = r;
```

```
        lcm = (lcm * p[i])/a;
    }
    return lcm;

int main(){
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], rem[n];

    printf("Enter the ~~number~~ CPU burst time
        : \n");
    for (int i = 0; i<n; i++) scanf("%d", &b[i]);
    printf("Enter time periods: \n")

    double sum = 0.0;
    for (int i = 0; i<n; i++){
        sum += (double) b[i]/pt[i];
    }

    double rhs = n * (pow(2.0,(1.0/n)) - 1.0);
    if sum( > rhs){
        printf("The given set of processes is
            not schedulable. \n")
        exit(0);
    }

    while (time<1){
        int task to Execute = -1;
        for (int i = 0; i<n; i++){
            if (time == nextRelease[i])
            rem[i] = b[i];
            nextRelease[i] += pt[i];
```

```
if (taskToExecute != -1) {
    if (prev != taskToExecute) {
        printf("%.dmp: Task %d is running.\n",
                time, proc[taskToExecute]);
        prev = TaskToExecute.
    }
    return 0;
}
```

o/p

Enter number of process : 3
Enter CPU burst times :
2 1 2
Enter time periods :
5 3 7
Rate Monotonic Scheduling :

PID	Burst	Period
2	1	3
1	2	5
3	2	7

1.019048 <= 0.779763 => false
Not schedulable.

4b Earliest deadline first

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

void sort (int proc[], int d[], int b[],
        int pt[], int n)

int temp=0;
for (int i=0; i<n ; i++)
{
  for (int j=i; j<n ; j++)
  {
    if (d[j] < d[i])
    {
      temp = d[j];
      d[j] = d[i];
      d[i] = temp;
      temp = pt[i];
      pt[i] = pt[j]
      pt[j] = temp ;
      temp = b[j];
      b[j] = b[i];
      b[j] = temp;
    }
  }
}
}
```

```
int gcd (int a, int b) {
    int r;
    while (b>0)
    {   r = a % b;
        a = b;
        b = r;
    }
    return a;
}

int lcm (int p[], int n)
{
    int lcm = p[0];
    for (int i = 1; i<n; i++)
        lcm = (lcm * p[i]) / gcd (lcm, p[i]);
    return lcm;
}

void main ()
{
    int n;
    printf ("Enter the number of processes: ")
    scanf ("%d", &n);
    int proc[n], b[n], pt[n], d[n], rem[n]
    printf ("Enter CPU burst times: \n")
    for (int i=0; i<n; i++) {
        scanf ("%d", &b[i]),
        rem[i] = b[i];
    }
    printf ("Enter the deadlines : \n");
    for (int i=0 ; i<n; i++)
        scanf ("%d", &d[i]);
```

```
bot (proc, d, b, pt, n);
int l = lcmul (pt, n);
printf("\n Earliest Deadline scheduling:
                    \n");
printf("PID \t Burst \t Deadline \t
                    Period \n");
int time = 0, prev = 0, x = 0;
int nextDeadline[n];
for (int i = 0 ; i < n ; i++)
{
    nextDeadline[i] = d[i];
    rem[i] = b[i];
}
while (time < l) {
  for (int i = 0 ; i < n ; i++)
  {  if (time % pt[i] == 0 && time != 0)

      nextDeadline[i] = time + d[i];
      rem[i] = b[i];
    }

  }

int minDeadline = l + 1;
int taskToExecute = -1;
for (int i = 0 ; i < n ; i++)
{

  if (rem[i] > 0 && next
      minDeadline > nextDeadline[i]
      taskToExecute = i;
    }
  }
  time++
}
}
```

O/P

Enter number of process : 3

Enter CPU burst times :

2 1 2

Enter deadline :

5 3 7

Enter time periods :

5 3 7

Earliest Deadline scheduling :

PID	Burst	Deadline	Period
2	1	3	3
1	2	5	5
3	2	7	7

Scheduling occurs for 105 ms.

5/6

P.T.O

5/08/24

4c **Proportional scheduling**

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

void sort_by_burst (int proc[], int burst[],
    int tickets[], int remaining[], int n)
{
  for (int i=0 ; i<n-1 ; i++) {
    for (int j=0 ; j<n-i-1 ; j++) {
      if (burst[j] > burst[j+1]) {
        int temp = proc[j];
        proc[j] = proc[j+1];
        proc[j+1] = temp;

        temp = burst[j];
        burst[j] = burst[j+1];
        burst[j+1] = temp;

        temp = tickets[j];
        tickets[j] = tickets[j+1];
        tickets[j+1] = temp;

      }
    }
  }
}

int main() {
  int n;
  printf("Enter number of processes: ");
  scanf("%d", &n);
```

```
printf("Enter the CPU burst times & number of
        tickets for each process: \n");
for(int i=0; i<n; i++){
    proc[i] = i+1;
    print("Process %d burst time: ", i+1);
    scanf("%d", &burst[i]);
    printf("Process %d tickets: ", i+1);
    scanf("%d", &tickets[i]);
    remaining[i] = burst[i];
    total_tickets += tickets[i];
}
sort_by_burst(proc, burst, tickets, remaining);
printf("\n Proportional share scheduling
        (Lottery Scheduling): \n");
printf("PID\t Burst\t Tickets \n");
for(int i=0; i<n; i++){
    printf("%d \t %d \t %d \n", proc[i],
                burst[i], tickets[i]);
}
int time 20
while(1){
    int active_process 20;
    for(int i=0; i<n; i++){
        if (remaining[i] > 0){
            active_processes ++;
        }
    }
    if (active_process == 0){
        break;
    }
```

Date
Page 14

8 4 3 4
9 4 1 4
10 4 3 4
11 4 CPU is idle
12 4 4
13 4 4 4
14 4 4
15 4 4
16 4 4
17 Process 1 is running
18 CPU is idle
19 Process 1 is running
20 CPU is idle
21 4
22 4
23 4
24 4
25 4
26 4
27 4
28 Process 1 is running
29 ~~Process~~ 1

5/6

```
int winning_ticket = rand() % total_tickets;
int cumulative_tickets = 0;
int selected_process = -1;

for (int i=0; i<n; i++) {
    if (remaining[i] > 0) {
        cumulative_tickets += tickets[i];
        if (winning_ticket < cumulative_tickets)
        {
            selected_process = i;
            break;
        }
    }
}
}
printf("Scheduling completed.\n");
return 0;
}
```

O/P

proportional share scheduling

PID	Burst	Tickets
2	3	2
1	5	3
3	7	4

ons	process 2 is running		
1	1	3	1
2	1	2	0
3	1	3	1
4	1	3	1
5	1	3	1
6	1	2	1
7		3	

12/06/24

Q5 # Producer-Consumer using semaphore

```
#include <stdio.h>
#include <stdlib.h>
int mutex =1, full=0, empty=7, x=0;
void produce(){
    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 item %d", x);
    x--;
    mutex = signal (mutex);
}

int wait (int s) {
    return (--s);
}
int signal (int s) {
    return (++s);
}
```

P.T.O.

```
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 :
                printf("\n Number of items remaining
                        in buffer : %d \n", x);
                exit(0);
                break;
        }
    }
    return 0
}
```

O/P

1. Producer
2. Consumer
3. Exit

Enter your choice : 1
Producer produces item 1
Enter your choice : 1
Producer produces item 2

⋮

Enter your choice : 1
Producer produces item 7
Enter your choice : 1
Buffer is full !!
Enter your choice : 2
Consumer consumes item 7
Enter your choice : 2
Consumer consumes item 6
Enter your choice : 3
Number of items remaining in buffer : 5

Q6 # Dining Philosopher problem.

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

#define MAX_PHILOSOPHERS 10
# define MAX_CHOPSTICKS MAX_PHILOSOPHERS

void dine (int phil_position);
pthread_t philosophers [MAX_PHILOSOPHERS];
pthread_mutex_t chopsticks [MAX_CHOPSTICKS];

int main() {
    int num_total_philosophers, num_hungry_philosophers,
        i, status;
    int philosophers_positions [MAX_PHILOSOPHERS];
    printf ("Enter the total number of philosophers
    (maximum %d): ", MAX_PHILOSOPHERS);
    scanf ("%d", & num_total_philosophers);
    if (num_total_philosophers < 2 || num_total_
    philosophers > MAX_PHILOSOPHERS) {
        printf (" Invalid number of philosophers. Must
        between 2 & %d. \n", MAX_PHILOSOPHERS);
        return 1;
    }
    printf (" Enter the number of philosophers who are
    hungry (maximum %d): ", num_total_philosophers);
    scanf ("%d", & num_hungry_philosophers);
```

PTO

```
if (num_hungry_philosophers <2 || num_
hungry_philosopher > num_total_philosophers){
printf ("Invalid number of hungry philosophers
Must be b/w 2 & %d \n", num_total_phil);

for (i=0; i<num_hungry_philosophers; i++){
scanf ("%d", &philosopher_positions[i]);
if (philosopher_position[i] <1 || philosopher_
position[i] > num_total_philosophers){
printf ("Invalid position for philosopher
%d. Position must be b/w 1 & %d
\n", i+1, num_total_philosophers);
return 1;
}
}

void dine (int phil_position) {
printf ("\n Philosopher %d is thinking \n",
phil_position);
pthread_mutex_lock (&chopsticks [phil_pos-1]);
pthread_mutex_lock (&chopsticks [phil_position
% MAX_CHOPSTICKS]);
printf ("Philosopher %d is eating \n", phil_
position);
sleep(3);
pthread_mutex_unlock (&chopsticks [phil_
position -1]);
pthread_mutex_unlock (&chopsticks [phil_position
% MAX_CHOPSTICKS
printf ("Philosopher %d finished eating \n",
phil_position);
}
```

O/P

Enter total number of philosophers : 10
Enter total hungry philosophers : 3
Enter position of 3 hungry philosopher : 2 4 5

philosophers 5 is thinking
philosophers 5 is eating

Philosopher 2 is thinking
Philosopher 2 is eating

Philosopher 4 is thinking
Philosopher 2 finished eating
Philosopher 4 is eating
Philosopher 5 finished eating
Philosopher 4 finished eating

19/06/24

Q7# Banker's Algorithm

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

# define MAX_PROCESS 10
# define MAX_RESOURCES 10

void calculateNeed(int need[MAX_PROCESS]
[MAX_RESOURCES], int max[MAX_PROCESS][MAX_
RESOURCES], int allot[MAX_PROCESS][MAX_RESOURCES]
){
  for (int i = 0; i< np; i++)
     for (int j = 0; j<nr; j++)
        need [i][j] = max[i][j] - allot[i][j];
}

bool isSafe (int process[], int avail[], int max[]
[MAX_RESOURCES], int allot[][MAX_RESOURCES],
int np, int nr){
  int need[MAX_PROCESS][MAX_RESOURCES];
  calculateNeed(need, max, allot, np, nr);

  int count=0;
  while (count<np){
    bool found= false;
    for (int p=0; p<np; p++){
      if (finish[p] ==0){
        int j;
        for (j=0; j<nr; j++)
          if (need[p][j] > work[j])
            break;
        if (j==nr){
```

```
            for(int k=0; k<nr; k++)
                work[k] += allot[p][k];
            safeseq[count++] = p;
            finish[p] = 1;
            found = true;
        }
    }
}
if (found == false) {
    printf("System is not in safe state\n");
    return false;
}
}
int main() {
    int np, nr;
    printf("Enter number of processes: ");
    scanf("%d", &np);
    printf("Enter number of resource type");
    scanf("%d", &nr);
    int process[MAX_PROCESSES];
    for(int i=0; i<np; i++) process[i] = 1;

    int avail[MAX_RESOURCES];
    printf("Enter available resources:");
    for (int i=0; i<nr; i++)
        scanf(" %d", &avail[i]);

    int allot[MAX_PROCESS][MAX_RESOURCES];
    printf("Enter allocation resource matrix:\n");
    for (int i=0; i<np; i++) {
        printf("Process %d: ", i);
        for (int j=0; j<nr; j++)
            scanf("%d", &allot[i][j]);
```

```
inSafe (process, avail, max, allot, np, nr);
return 0;
}
```

o/p

```
Enter number of process : 5
Enter number of resource type : 3
Enter available resource : 3 3 2
Enter maximum resource Matrix:
Process 0 : 7 5 3
Process 1 : 3 2 2
Process 2 : 9 0 2
Process 3 : 2 2 2
Process 4 : 4 3 3
Enter allocation resource matrix:
Process 0 : 0 1 0
Process 1 : 2 0 0
Process 2 : 3 0 2
Process 3 : 2 1 1
Process 4 : 0 0 2
System in safe state
Safe Sequence : 1 3 4 0 2
```

19/6

Q: # Deadlock detection

```
# include <stdio.h>

# define MAX_PROCESSES 10
# define MAX_RESOURCES 10

int allocation [MAX_PROCESSES] [MAX_RESOURCES];
int max need [MAX_PROCESSES] [MAX_RESOURCES];
int available [MAX_RESOURCES];
int n_processes, n_resources;

void initialise ()
{
  int i,j;
  printf ("Enter number of processes : ");
  scanf ("%d", & n_processes);
  printf ("%d", &n_resources);
  printf ("Enter allocation matrix (%d x %d) : \n",
          n_processes, n_resources);
  for (i=0; j<n_resources; j++)
  {
    scanf ("%d", & allocation [i][j]);
  }
printf ("Enter maxim need matrix (%d x %d) : \
        n_processes, n_resources);
for (i=0; i<n_processes, i++) {
  for (j=0; j<n_resources; j++)
  {
    scanf ("%d", &max_need[i][j]);
  }
}
}
```

```
void detect deadlock()
{
  int i, j, k;
  int work[MAX-RESOURCES];
  int finish[MAX_PROCESSES];
  int safe_sequence[MAX_PROCESSES];
  int safe_sequence_count = 0;
  for (j=0; j<n_resources; j++)
  {
     work[j] = available[j];
  }
  for (i=0; i<n_processes; i++)
  {
     finish[i] = 0;
  }
  int found;
  do {
    found = 0
    for (i=0; i<n_processes; i++){
      if (!finish[i]){
        int can_allocate = 1;
        for (j=0; j<n_resources; j++)
        {
          if (max_need[i][j] - allocation[i][j]
                                  > work[j])
          {
            can_allocate = 0;
            break;
          }
        }
      }
      if (canallocate)
      {
        for (j=0; j<n_resources; j++)
        {
          work[j] += allocation[i][j];
        }
```

```
        finish[i] = 1;
        safe_sequence[safe_sequence_counter]=i;
        found=1
      }
    }
  }
  printf("No deadlock detected, In safe sequence:");
  for(i=0; i<n_processes; i++)
  {
    printf("%d", safe_sequence[i]);
  }
  printf("\n");
}
int main()
{
    initialise();
    detect_deadlock();
    return 0;
}
```

O/P

```
Enter no. of process : 5
Enter no. of resources : 3
Enter allocation matrix (5x3):
010
200
303
211
002
Enter maximum need matrix (5x3):
000
202
000
100
002
Available
000
```

No deadlock detected
safe sequence : 0 2 3 4 1

3/07/24

Q9 # Contiguous memory allocation

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

# define MAX 25

void firstFit (int nb, int nf, int b[], int f[]){
  int ff[MAX] = {0};
  int allocate [MAX] = {0};
  for (int i = 0 ; j<nf, i++) {
    ff[i] = -1;
    for (int j = 0; j<nb ; j++) {
      if (allocated[j] == 0 && b[j] >= f[i]) {
        ff[i] = j;
        allocated [j] = 1;
        break;
      }
    }
  }
  printf("\n File_no: \t File_size : \t Block_no \t Block_size:\t");
  for (int i=0 ; i<nf, i++){
    if (ff[i] != -1 )
      printf("\n %d \t %d \t %d \t\t - \t\t %d", i+1, f[i],
                                    ff[i]+1, b[ff[i]],
    else
      printf("\n %d \t\t %d \t\t - \t\t -", i+1, f[i]),
  }
}
```

RTO

```
void bestfit(int nb, int nf, int b[], int f[]){
int ff[MAX]={0};
int allocated[MAX]={0};
for(int i=0; i<nf; i++){
  int best=-1;
  ff[i]=-1;
  for(int j=0; j<nb; j++){
    if(allocated[j]==0 && b[j]>=f[i]){
      if(best==-1 || b[j]<b[best])
        best=j;
    }
  }
  if(best!=-1){
   ff[i]=best;
   allocated[best]=-1;
  }
}
printf("\n File_no:\t File_size:\t Block_no:\t Block_size:");
for(int i=0; i<nf; i++){
  if(ff[i]!=-1)
    printf("\n%d\t\t%d\t\t%d\t\t%d", i+1, f[i], ff[i]+1
                                        b[ff[i]]);
  else
    printf("\n%d\t\t%d\t\t-\t\t-", i+1, f[i]);
  }
}
```

PTO

```
int main() {
 int nb, nf, choice;
 printf ("Memory Management scheme");
 printf("\n Enter the number of blocks");
 scanf("%d", &nb);
 printf("Enter the number of files :");
 scanf("%d", &nf);
 int b[nb], f[nf];
 printf("\n Enter the size of blocks : \n");
 for(int i=0; i<nb ; i++){
  printf("Block %d", i+1);
  scanf("%d", &b[i]);
 }
 printf("Enter the size of files : \n");
 for(int i=0; i<nf ; i++){
  printf("file %d:", i+1);
  scanf("%d", &f[i]);
 }
 while (1){
 printf("\n 1. First fit \n 2. Best fit \n 3. Worst fit
                                    \n 4. Exit \n");
 printf(" Enter your choice :");
 scanf("%d", &choice);
 ~~scanf("%d"~~
 switch (choice){
  case 1:
   printf("\n\t Memory Management scheme -
                                  first fit \n");
   firstfit(nb, nf, b, f);
   break;
  case 2:
   printf("\n\t Memory M scheme - Best fit \n");
   bestfit(nb, nf, b, f);
   break;
```

```
case 3:
  printf("\n\t Memory M scheme - Best fit \n");
  bestfit(nb, nf, b, f);
  break;
case 4:
  printf("\n Exiting... \n");
  exit(0);
  break;
default:
   printf("\n Invalid choice. \n");
   break;
}
}
return 0;
}
```

O/P

Memory Management scheme
Enter the number of blocks : 5
Enter the number of files : 4
Enter the size of the blocks :
Block 1 : 500
Block 2 : 250
Block 3 : 350
Block 4 : 100
Block 5 : 150
Enter the size of files :
File 1 : 320
File 2 : 180
File 3 : 100
File 4 : 450
Enter your choice : 1
Memory management scheme - first fit

PTO

First fit

File no:	File-size:	Block no.	Block size:
1	320	1	500
2	150	2	250
3	100	3	350
4	450	—	—

Q10 # Page replacement algorithm:

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

void fifo (int f[], int n, int p[], int m)
{
  int index = 0;
  int p_fault = 0;
  for (int i=0; i<m; i++)
  {
    bool found = false;
    for (int j=0; j<n; j++)
    {
      if (f[j] == p[i])
      {
        found = true;
        break;
      }
    }
    if(!found)
    {
      f[index] = p[i];
      index = (index+1)%n;
      p_fault++;
    }
  }
  printf("FIFO Page faults : %d \n", p_fault);
}
```

P.T.O

```
void lru (int F[], int n, int P[], int m)
{
  int LU[n];
  int p_fault=0;
  int count=0;
  for (int i=0; i<n; i++)
  {
    F[i]=-1;
    LU[i]=-1;
  }
  for (int i=0; i<m; i++)
  {
    int j;
    bool found = false;
    for (j=0; j<n; j++)
    {
      if (F[j] == P[i])
      {
        found = true;
        LU[j] = count++;
        break;
      }
    }
    if (!found)
    {
      int lru_index=0
      for (j=1; j<n; j++)
      {
        if (LU[j] < LU[lru_index])
        {
          lru_index = j;
        }
      }
```

```
            f[lru_index] = P[i];
            LU[lru_index] = count++;
            p_fault++;
        }
    }
    printf("LRU Page Faults : %d \n", p_fault);
}

void optimal(int f[], int n, int P[], int m)
{
    int p_fault=0;
    for (int i=0; i<n; i++)
    {
        f[i] = -1;
    }
    for (int i=0; i<m; i++)
    {
        bool found = false;
        for (int j=0; j<n; j++)
        {
            if (f[j] == P[i])
            {
                found = true;
                break;
            }
        }
        if (!found)
        {
            int replace_i = -1;
            int far = i+1;
```

P.T.O

```
for(int j=0; j<n; j++)
{
  int k;
  for(k=i+1; k<m; k++)
  {
    if(F[j] == P[k])
    {
      if(k>for)
      {
        for=k;
        replace_i=j;
      }
      break;
    }
  }
  if(k==m)
  {
    replace_i=j;
    break;
  }
}
if(replace_i==-1)
{
  replace_i=0;
}
F[replace_i]=P[i];
p_fault++;
}
}
printf("Optimal page faults: %d\n",
                        p_fault);
}
```

```
int main()
{
    int n;
    int m;
    printf("Enter the number of Frames: ");
    scanf("%d", &n);
    printf("Enter the length of reference string: ");
    scanf("%d", &m);
    int P[m];
    printf("Enter the reference string: ");
    for (int i=0 ; i<m ; i++)
    {
        scanf("%d", &P[i]);
    }
    int F[n];
    for (int i=0; i<n ; i++)
    {
        F[i] = -1;
    }
    fifo (F, n, P, m);
    lru (F, n, P, m);
    optimal (F, n, P, m);
    return 0;
}
```

O/P

Enter the number of frames: 3
Enter the length of reference strings: 20
Enter the reference string: 0 9 0 1 8 18 78 7 12
8 2 7 8 2 3 8 3

FIFO page faults: 8
LRU page faults: 9
Optimal page faults: 7

10/7 completed