

1. Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

→ FCFS

→ SJF (pre-emptive & Non-preemptive)

Program:

a. FCFS

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

```
#define MAX_PROCESS 10
```

```
void fcfs(int n, int at[], int bt[]) {
```

```
    int ct[MAX_PROCESS];
```

```
    int tat[MAX_PROCESS];
```

```
    int wt[MAX_PROCESS];
```

```
    int total_wt = 0;
```

```
    int total_tat = 0;
```

```
    int current_time = 0;
```

```
    // Initialize completion time array with -1
```

```
    for (int i = 0; i < n; i++) {
```

```
        ct[i] = -1;
```

```
}
```

```
    // Find completion time for each process
```

```
    for (int i = 0; i < n; i++) {
```

```
        if (current_time < at[i]) {
```

```
            current_time = at[i];
```

```
}
```

```

        ct[i] = current_time + bt[i];
        current_time = ct[i];
    }

// Find turnaround time for each process

for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    total_tat += tat[i];
}

// Find waiting time for each process

for (int i = 0; i < n; i++) {
    wt[i] = tat[i] - bt[i];
    total_wt += wt[i];
}

// Print the results

printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");
for (int i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
}

printf("\nAverage waiting time: %.2f", (float)total_wt / n);
printf("\nAverage turnaround time: %.2f", (float)total_tat / n);

}

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

```

```

int at[n], bt[n];

printf("Enter the arrival time:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &at[i]);
}

printf("Enter the burst time:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &bt[i]);
}

fcfs(n, at, bt);

return 0;
}

```

Output:

```

Enter the number of processes: 4
Enter the arrival time:
0
1
5
6
Enter the burst time:
2
2
3
4

Process  Arrival Time    Burst Time     Completion Time Turnaround Time Waiting Time
P1      0            2              2                  2                      0
P2      1            2              4                  3                      1
P3      5            3              8                  3                      0
P4      6            4             12                 6                      2

Average waiting time: 0.75
Average turnaround time: 3.50
Process returned 0 (0x0)  execution time : 16.964 s
Press any key to continue.

```

b. SJF (pre-emptive)

```
#include <stdio.h>

#define MAX 10

void sjf_preemptive(int n, int at[], int bt[]) {

    int ct[MAX];
    int tat[MAX];
    int wt[MAX];
    int rt[MAX];
    int total_wt = 0;
    int total_tat = 0;
    int completed = 0;
    int current_time = 0;
    int is_completed[MAX] = {0};

    for (int i = 0; i < n; i++) {
        rt[i] = bt[i];
    }

    while (completed < n) {
        int shortest_job = -1;
        int min_bt = 9999;

        for (int i = 0; i < n; i++) {
            if (at[i] <= current_time && rt[i] < min_bt && rt[i] > 0) {
                shortest_job = i;
                min_bt = rt[i];
            }
        }

        if (shortest_job == -1) {
            current_time++;
            continue;
        }

        rt[shortest_job]--;
        if (rt[shortest_job] == 0) {
```

```

        completed++;

ct[shortest_job] = current_time + 1;

tat[shortest_job] = ct[shortest_job] - at[shortest_job];

total_tat += tat[shortest_job];

wt[shortest_job] = tat[shortest_job] - bt[shortest_job];

if (wt[shortest_job] < 0) wt[shortest_job] = 0;

total_wt += wt[shortest_job];

is_completed[shortest_job] = 1;

}

current_time++;

}

printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

for (int i = 0; i < n; i++) {

printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);

}

printf("\nAverage waiting time: %.2f", (float)total_wt / n);

printf("\nAverage turnaround time: %.2f", (float)total_tat / n);

}

int main() {

int n, i;

printf("Enter the number of processes: ");

scanf("%d", &n);

int at[n], bt[n];

printf("Enter the arrival time:\n");

for (i = 0; i < n; i++) {

scanf("%d", &at[i]);

}

printf("Enter the burst time:\n");

for (i = 0; i < n; i++) {

scanf("%d", &bt[i]);
}

```

```

    }
    sjf_preemptive(n, at, bt);
    return 0;
}

```

Output:

```

Enter the number of processes: 5
Enter the arrival time:
2 1 4 0 2
Enter the burst time:
1 5 1 6 3

Process Arrival Time      Burst Time      Completion Time Turnaround Time Waiting Time
1      2                  1                3                  1                  0
2      1                  5                11                 10                 5
3      4                  1                5                  1                  0
4      0                  6                16                 16                 10
5      2                  3                7                  5                  2

Average waiting time: 3.40
Average turnaround time: 6.60
Process returned 0 (0x0)  execution time : 21.791 s
Press any key to continue.

```

c.SJF (Non-preemptive)

```

#include <stdio.h>

#define MAX 10

void sjf_non_preemptive(int n, int at[], int bt[]) {
    int ct[MAX];
    int tat[MAX];
    int wt[MAX];
    int rt[MAX];
    int total_wt = 0;
    int total_tat = 0;
    int completed = 0;
    int current_time = 0;
    int shortest_job = 0;
    int min_bt = 9999;
    int is_completed[MAX] = {0};

```

```

for (int i = 0; i < n; i++) {
    rt[i] = bt[i];
}

while (completed < n) {
    for (int i = 0; i < n; i++) {
        if (at[i] <= current_time && rt[i] < min_bt && !is_completed[i]) {
            shortest_job = i;
            min_bt = rt[i];
        }
    }

    rt[shortest_job]--;
    if (rt[shortest_job] == 0) {
        completed++;
        min_bt = 9999;
        is_completed[shortest_job] = 1;
        ct[shortest_job] = current_time + 1;
        tat[shortest_job] = ct[shortest_job] - at[shortest_job];
        total_tat += tat[shortest_job];
        wt[shortest_job] = tat[shortest_job] - bt[shortest_job];
        if (wt[shortest_job] < 0) wt[shortest_job] = 0;
        total_wt += wt[shortest_job];
    }
    current_time++;
}

printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");
for (int i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
}

printf("\nAverage waiting time: %.2f", (float)total_wt / n);
printf("\nAverage turnaround time: %.2f", (float)total_tat / n);

```

```
}
```

```
int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[n], bt[n];
    printf("Enter the arrival time:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &at[i]);
    }
    printf("Enter the burst time:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
    }
    sjf_non_preemptive(n, at, bt);
    return 0;
}
```

Output:

```
Enter the number of processes: 4
Enter the arrival time:
0 0 0 0
Enter the burst time:
6 8 7 3

Process  Arrival Time    Burst Time    Completion Time Turnaround Time Waiting Time
1      0            6              9             9                 3
2      0            8              24            24                16
3      0            7              16            16                9
4      0            3              3              3                 0

Average waiting time: 7.00
Average turnaround time: 13.00
Process returned 0 (0x0)  execution time : 18.046 s
Press any key to continue.
```

2. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

→ Priority (pre-emptive & Non-pre-emptive)

→ Round Robin (Experiment with different quantum sizes for RR algorithm)

Program:

a. Priority (pre-emptive)

```
#include<stdio.h>

void sort (int proc_id[], int p[], int at[], int bt[], int b[], int n){

    int min = p[0], temp = 0;

    for (int i = 0; i < n; i++){

        min = p[i];

        for (int j = i; j < n; j++){

            if (p[j] < min){

                temp = at[i];

                at[i] = at[j];

                at[j] = temp;

                temp = bt[j];

                bt[j] = bt[i];

                bt[i] = temp;

                temp = b[j];

                b[j] = b[i];

                b[i] = temp;

                temp = p[j];

                p[j] = p[i];

                p[i] = temp;

                temp = proc_id[i];

                proc_id[i] = proc_id[j];

                proc_id[j] = temp;

            }

        }

    }

}
```

```

}

void main (){
    int n, c = 0;
    printf ("Enter number of processes: ");
    scanf ("%d", &n);
    int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], m[n], b[n], rt[n], p[n];
    double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
    for (int i = 0; i < n; i++){
        proc_id[i] = i + 1;
        m[i] = 0;
    }
    printf ("Enter priorities:\n");
    for (int i = 0; i < n; i++)
        scanf ("%d", &p[i]);
    printf ("Enter arrival times:\n");
    for (int i = 0; i < n; i++)
        scanf ("%d", &at[i]);
    printf ("Enter burst times:\n");
    for (int i = 0; i < n; i++){
        scanf ("%d", &bt[i]);
        b[i] = bt[i];
        m[i] = -1;
        rt[i] = -1;
    }
    sort(proc_id, p, at, bt, b, n);
    int count = 0, pro = 0, priority = p[0];
    int x = 0;
    c = 0;
    while (count < n){
        for (int i = 0; i < n; i++){
            if (at[i] <= c && p[i] >= priority && b[i] > 0 && m[i] != 1){

```

```

        x = i;
        priority = p[i];
    }

}

if (b[x] > 0){

    if (rt[x] == -1)

        rt[x] = c - at[x];

    b[x]--;
    c++;
}

if (b[x] == 0){

    count++;

    ct[x] = c;

    m[x] = 1;

    while (x >= 1 && b[x] == 0)

        priority = p[--x];
}

if (count == n)

    break;
}

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];

printf ("Priority scheduling(Pre-Emptive):\n");

printf ("PID\tPrior\tAT\tBT\tCT\tTAT\tWT\tRT\n");

for (int i = 0; i < n; i++)

    printf ("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc_id[i], p[i], at[i],
           bt[i], ct[i], tat[i], wt[i], rt[i]);

for (int i = 0; i < n; i++){

    ttat += tat[i];
}

```

```

        twt += wt[i];

    }

avg_tat = ttat / (double) n;
avg_wt = twt / (double) n;
printf ("\nAverage turnaround time:%lfms\n", avg_tat);
printf ("\nAverage waiting time:%lfms\n", avg_wt);

}

```

Output:

```

Enter number of processes: 4
Enter priorities:
10 20 30 40
Enter arrival times:
0 1 2 4
Enter burst times:
5 4 2 1
Priority scheduling(Pre-Emptive):
PID    Prior    AT      BT      CT      TAT      WT      RT
P1      10       0       5       12      12       7       0
P2      20       1       4       8       7       3       0
P3      30       2       2       4       2       0       0
P4      40       4       1       5       1       0       0

Average turnaround time:5.500000ms
Average waiting time:2.500000ms
Process returned 33 (0x21)  execution time : 22.246 s
Press any key to continue.

```

b. Priority (Non-pre-emptive)

```

#include<stdio.h>

void sort (int proc_id[], int p[], int at[], int bt[], int n){

int min = p[0], temp = 0;
for (int i = 0; i < n; i++)
{
    min = p[i];
    for (int j = i; j < n; j++)
    {
        if (p[j] < min)

```

```

    {
        temp = at[i];
        at[i] = at[j];
        at[j] = temp;
        temp = bt[j];
        bt[j] = bt[i];
        bt[i] = temp;
        temp = p[j];
        p[j] = p[i];
        p[i] = temp;
        temp = proc_id[i];
        proc_id[i] = proc_id[j];
        proc_id[j] = temp;
    }
}

void main (){
    int n, c = 0;
    printf ("Enter number of processes: ");
    scanf ("%d", &n);
    int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], m[n], rt[n], p[n];
    double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
    for (int i = 0; i < n; i++){
        proc_id[i] = i + 1;
        m[i] = 0;
    }
    printf ("Enter priorities:\n");
    for (int i = 0; i < n; i++)
        scanf ("%d", &p[i]);
}

```

```

printf ("Enter arrival times:\n");
for (int i = 0; i < n; i++)
    scanf ("%d", &at[i]);
printf ("Enter burst times:\n");
for (int i = 0; i < n; i++)
{
    scanf ("%d", &bt[i]);
    m[i] = -1;
    rt[i] = -1;
}
sort (proc_id, p, at, bt, n);
int count = 0, pro = 0, priority = p[0];
int x = 0;
c = 0;
while (count < n){
    for (int i = 0; i < n; i++){
        if (at[i] <= c && p[i] >= priority && m[i] != 1){
            x = i;
            priority = p[i];
        }
    }
    if (rt[x] == -1)
        rt[x] = c - at[x];
    if (at[x] <= c)
        c += bt[x];
    else
        c += at[x] - c + bt[x];
    count++;
    ct[x] = c;
    m[x] = 1;
    while (x >= 1 && m[--x] != 1)  {

```

```

priority = p[x];
break;
}
x++;
if (count == n)
break;
}

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];
printf ("\nPriority scheduling:\n");
printf ("PID\tPrior\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++)
printf ("P%d\t %d\t%d\t%d\t%d\t%d\t%d\n", proc_id[i], p[i], at[i],
bt[i], ct[i], tat[i], wt[i], rt[i]);
for (int i = 0; i < n; i++){
ttat += tat[i];
twt += wt[i];
}
avg_tat = ttat / (double) n;
avg_wt = twt / (double) n;
printf ("\nAverage turnaround time:%lfms\n", avg_tat);
printf ("\nAverage waiting time:%lfms\n", avg_wt);
}

```

Output:

```

Enter number of processes: 4
Enter priorities:
10 20 30 40
Enter arrival times:
0 1 2 4
Enter burst times:
5 4 2 1

Priority scheduling:
PID Prior AT BT CT TAT WT RT
P1 10 0 5 5 5 0 0
P2 20 1 4 12 11 7 7
P3 30 2 2 8 6 4 4
P4 40 4 1 6 2 1 1

Average turnaround time:6.000000ms
Average waiting time:3.000000ms
Process returned 33 (0x21) execution time : 22.748 s
Press any key to continue.
-
```

c. Round Robin (Experiment with different quantum sizes for RR algorithm)

```

#include <stdio.h>

#define MAX 10

void round_robin(int n, int bt[], int quantum) {

    int wt[MAX] = {0};
    int tat[MAX] = {0};
    int remaining_bt[MAX];
    int total_wt = 0, total_tat = 0;
    int time = 0;

    for (int i = 0; i < n; i++) {
        remaining_bt[i] = bt[i];
    }

    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_bt[i] > 0) {
                done = 0;
                if (remaining_bt[i] > quantum) {
                    time += quantum;

```

```

        remaining_bt[i] -= quantum;

    } else {
        time += remaining_bt[i];
        wt[i] = time - bt[i];
        remaining_bt[i] = 0;
    }
}

if (done == 1) break;
}

for (int i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    total_wt += wt[i];
    total_tat += tat[i];
}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
}

printf("\nAverage waiting time: %.2f", (float)total_wt / n);
printf("\nAverage turnaround time: %.2f", (float)total_tat / n);
}

int main() {
    int n, quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int bt[MAX];
    printf("Enter Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
    }
}

```

```

        scanf("%d", &bt[i]);
    }

    printf("Enter the size of time slice (quantum): ");

    scanf("%d", &quantum);

    round_robin(n, bt, quantum);

    return 0;
}

```

Output:

```

Enter the number of processes: 3
Enter Burst Time for each process:
Process 1: 24
Process 2: 3
Process 3: 3
Enter the size of time slice (quantum): 3

Process  Burst Time      Waiting Time     Turnaround Time
1          24              6                  30
2          3                3                  6
3          3                6                  9

Average waiting time: 5.00
Average turnaround time: 15.00
Process returned 0 (0x0)  execution time : 19.434 s
Press any key to continue.

```

3. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

Program:

```

#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int at[], int wt[]) {
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        wt[i] = bt[i-1] + wt[i-1] - at[i-1];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}

```

```

        wt[i] = 0;
    }

}

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

void roundRobin(int processes[], int n, int bt[], int at[], int quantum) {
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    int remaining_bt[n];
    int completed = 0;
    int time = 0;
    for (int i = 0; i < n; i++) {
        remaining_bt[i] = bt[i];
    }
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_bt[i] > 0 && at[i] <= time) {
                if (remaining_bt[i] <= quantum) {
                    time += remaining_bt[i];
                    remaining_bt[i] = 0;
                    ct[i] = time;
                    completed++;
                } else {
                    time += quantum;
                    remaining_bt[i] -= quantum;
                }
            }
        }
    }
}

```

```

findWaitingTime(processes, n, bt, at, wt);

findTurnaroundTime(processes, n, bt, wt, tat);

printf("Processes Burst Time Arrival Time Waiting Time Turnaround Time Completion Time\n");

for (int i = 0; i < n; i++) {

    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], bt[i], at[i], wt[i], tat[i], ct[i]);

    total_wt += wt[i];

    total_tat += tat[i];

}

printf("Average Waiting Time (Round Robin) = %f\n", (float)total_wt / n);

printf("Average Turnaround Time (Round Robin) = %f\n", (float)total_tat / n);

}

void fcfs(int processes[], int n, int bt[], int at[]) {

    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, at, wt);

    findTurnaroundTime(processes, n, bt, wt, tat);

    printf("Processes Burst Time Arrival Time Waiting Time Turnaround Time Completion Time\n");

    for (int i = 0; i < n; i++) {

        ct[i] = at[i] + bt[i];

        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], bt[i], at[i], wt[i], tat[i], ct[i]);

        total_wt += wt[i];

        total_tat += tat[i];

    }

    printf("Average Waiting Time (FCFS) = %f\n", (float)total_wt / n);

    printf("Average Turnaround Time (FCFS) = %f\n", (float)total_tat / n);

}

int main() {

    int processes[] = {1, 2, 3, 4, 5};

    int n = sizeof(processes) / sizeof(processes[0]);

    int bt[] = {10, 5, 8, 12, 15};

    int at[] = {0, 1, 2, 3, 4};

    int quantum = 2;

```

```

    roundRobin(processes, n, bt, at, quantum);

    fcfs(processes, n, bt, at);

    return 0;

}

```

Output:

Processes	Burst Time	Arrival Time	Waiting Time	Turnaround Time	Completion Time
P1	10	0	0	10	39
P2	5	1	10	15	23
P3	8	2	14	22	33
P4	12	3	20	32	45
P5	15	4	29	44	50
Average Waiting Time (Round Robin) = 14.600000					
Average Turnaround Time (Round Robin) = 24.600000					
Processes	Burst Time	Arrival Time	Waiting Time	Turnaround Time	Completion Time
P1	10	0	0	10	10
P2	5	1	10	15	6
P3	8	2	14	22	10
P4	12	3	20	32	15
P5	15	4	29	44	19
Average Waiting Time (FCFS) = 14.600000					
Average Turnaround Time (FCFS) = 24.600000					
Process returned 0 (0x0) execution time : 0.141 s					
Press any key to continue.					

4. Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) Rate- Monotonic

b) Earliest-deadline First

c) Proportional scheduling

Program

a) Rate- Monotonic

```

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

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

    int temp = 0;

    for (int i = 0; i < n; i++){

        for (int j = i; j < n; j++){

            if (pt[j] < pt[i]){

                temp = pt[i];

                pt[i] = pt[j];

                pt[j] = temp;
            }
        }
    }
}

```

```

        pt[i] = pt[j];
        pt[j] = temp;
        temp = b[j];
        b[j] = b[i];
        b[i] = temp;
        temp = proc[i];
        proc[i] = proc[j];
        proc[j] = temp;
    }

}

}

int gcd (int a, int b){

    int r;

    while (b > 0){

        r = a % b;

        a = b;

        b = r;
    }

    return a;
}

int lcmul (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], rem[n];

printf ("Enter the CPU burst times:\n");

for (int i = 0; i < n; i++){

    scanf ("%d", &b[i]);

    rem[i] = b[i];

}

printf ("Enter the time periods:\n");

for (int i = 0; i < n; i++)

    scanf ("%d", &pt[i]);

for (int i = 0; i < n; i++)

    proc[i] = i + 1;

sort (proc, b, pt, n);

int l = lcmul (pt, n);

printf ("LCM=%d\n", l);

printf ("\nRate Monotone Scheduling:\n");

printf ("PID\t Burst\tPeriod\n");

for (int i = 0; i < n; i++)

    printf ("%d\t%d\t%d\n", proc[i], b[i], pt[i]);

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);

printf ("\n%lf <= %lf =>%s\n", sum, rhs, (sum <= rhs) ? "true" : "false");

if (sum > rhs)

    exit (0);

printf ("Scheduling occurs for %d ms\n\n", l);

int time = 0, prev = 0, x = 0;

while (time < l){

    int f = 0;

```

```

for (int i = 0; i < n; i++){
    if (time % pt[i] == 0)
        rem[i] = b[i];
    if (rem[i] > 0){
        if (prev != proc[i]){
            printf ("%dms onwards: Process %d running\n", time,
                    proc[i]);
            prev = proc[i];
        }
        rem[i]--;
        f = 1;
        break;
        x = 0;
    }
}

if (!f){
    if (x != 1){
        printf ("%dms onwards: CPU is idle\n", time);
        x = 1;
    }
}
time++;
}

```

Output:

```

Enter the number of processes:3
Enter the CPU burst times:
3 2 2
Enter the time periods:
20 5 10
LCM=20

Rate Monotone Scheduling:
PID      Burst    Period
2          2           5
3          2          10
1          3          20

0.750000 <= 0.779763 =>true
Scheduling occurs for 20 ms

0ms onwards: Process 2 running
2ms onwards: Process 3 running
4ms onwards: Process 1 running
5ms onwards: Process 2 running
7ms onwards: Process 1 running
8ms onwards: CPU is idle
10ms onwards: Process 2 running

Process returned 20 (0x14)  execution time : 22.670 s
Press any key to continue.

```

b) 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[i] = temp;
        temp = proc[i];
        proc[i] = proc[j];
        proc[j] = temp;
    }

}

}

int gcd (int a, int b){

    int r;

    while (b > 0){

        r = a % b;

        a = b;

        b = r;

    }

    return a;
}

int lcmul (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 the 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]);

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

for (int i = 0; i < n; i++)
    proc[i] = i + 1;

sort (proc, d, b, pt, n);

int l = lcmul (pt, n);

printf ("\nEarliest Deadline Scheduling:");
printf ("PID\t Burst\tDeadline\tPeriod\n");
for (int i = 0; i < n; i++)
    printf ("%d\t%d\t%d\t%d\n", proc[i], b[i], d[i], pt[i]);

printf ("Scheduling occurs for %d ms\n", l);

int time = 0, prev = 0, x = 0;
int nextDeadlines[n];
for (int i = 0; i < n; i++){
    nextDeadlines[i] = d[i];
    rem[i] = b[i];
}

while (time < l){
    for (int i = 0; i < n; i++){
        if (time % pt[i] == 0 && time != 0){
            nextDeadlines[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 && nextDeadlines[i] < minDeadline){

        minDeadline = nextDeadlines[i];

        taskToExecute = i;

    }

}

if (taskToExecute != -1){

    printf ("%dms : Task %d is running.\n", time, proc[taskToExecute]);

    rem[taskToExecute]--;

}

else{

    printf ("%dms: CPU is idle.\n", time);

}

time++;

}


```

Output:

```

Enter the CPU burst times:
0 1 2
Enter the deadlines:
8 5 4
Enter the time periods:
3 4 6

Earliest Deadline Scheduling:
PID      Burst    Deadline      Period
3          2        4            6
2          1        5            4
1          0        8            3
Scheduling occurs for 12 ms

0ms : Task 3 is running.
1ms : Task 3 is running.
2ms : Task 2 is running.
3ms: CPU is idle.
4ms : Task 2 is running.
5ms: CPU is idle.
6ms : Task 3 is running.
7ms : Task 3 is running.
8ms: CPU is idle.
9ms: CPU is idle.
10ms: CPU is idle.
11ms: CPU is idle.

Process returned 12 (0xC)  execution time : 18.265 s
Press any key to continue.

```

c) Proportional scheduling

```

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

#define MAX_TASKS 10
#define MAX_TICKETS 100
#define TIME_UNIT_DURATION_MS 100

struct Task {
    int tid;
    int tickets;
};

void schedule(struct Task tasks[], int num_tasks, int *time_span_ms) {
    int total_tickets = 0;
    for (int i = 0; i < num_tasks; i++) {
        total_tickets += tasks[i].tickets;
    }
}

```

```

}

srand(time(NULL));

int current_time = 0;

int completed_tasks = 0;

printf("Process Scheduling:\n");

while (completed_tasks < num_tasks) {

    int winning_ticket = rand() % total_tickets;

    int cumulative_tickets = 0;

    for (int i = 0; i < num_tasks; i++) {

        cumulative_tickets += tasks[i].tickets;

        if (winning_ticket < cumulative_tickets) {

            printf("Time %d-%d: Task %d is running\n", current_time, current_time + 1, tasks[i].tid);

            current_time++;

            break;
        }
    }

    completed_tasks++;
}

*time_span_ms = current_time * TIME_UNIT_DURATION_MS;
}

int main() {

    struct Task tasks[MAX_TASKS];

    int num_tasks;

    int time_span_ms;

    printf("Enter the number of tasks: ");

    scanf("%d", &num_tasks);

    if (num_tasks <= 0 || num_tasks > MAX_TASKS) {

        printf("Invalid number of tasks. Please enter a number between 1 and %d.\n", MAX_TASKS);

        return 1;
    }

    printf("Enter number of tickets for each task:\n");
}

```

```

for (int i = 0; i < num_tasks; i++) {
    tasks[i].tid = i + 1;
    printf("Task %d tickets: ", tasks[i].tid);
    scanf("%d", &tasks[i].tickets);
}
printf("\nRunning tasks:\n");
schedule(tasks, num_tasks, &time_span_ms);
printf("\nTime span of the Gantt chart: %d milliseconds\n", time_span_ms);
return 0;
}

```

Output:

```

Enter the number of tasks: 3
Enter number of tickets for each task:
Task 1 tickets: 10
Task 2 tickets: 20
Task 3 tickets: 30

Running tasks:
Process Scheduling:
Time 0-1: Task 3 is running
Time 1-2: Task 2 is running
Time 2-3: Task 1 is running

Time span of the Gantt chart: 300 milliseconds

Process returned 0 (0x0)   execution time : 18.093 s
Press any key to continue.

```

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

```

#include<stdio.h>

#include<stdlib.h>

int mutex = 1, full = 0, empty = 5, 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:
            printf("Program execution completed.\n");
            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);
    empty = wait(empty);

    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

Producer produces the item 4
Enter your choice:1

Producer produces the item 5
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 5
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:
2
Buffer is empty!!
Enter your choice:3
Program execution completed.
```

6. Write a C program to simulate the concept of Dining-Philosophers problem.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PHILOSOPHERS 5

void allow_one_to_eat(int hungry[], int n) {
    int isWaiting[MAX_PHILOSOPHERS];
    for (int i = 0; i < n; i++) {
        isWaiting[i] = 1;
    }
    for (int i = 0; i < n; i++) {
        printf("P %d is granted to eat\n", hungry[i]);
        isWaiting[hungry[i]] = 0;
        for (int j = 0; j < n; j++) {
            if (isWaiting[hungry[j]]) {
                printf("P %d is waiting\n", hungry[j]);
            }
        }
        for (int k = 0; k < n; k++) {
            isWaiting[k] = 1;
        }
        isWaiting[hungry[i]] = 0;
    }
}

void allow_two_to_eat(int hungry[], int n) {
    if (n < 2 || n > MAX_PHILOSOPHERS) {
        printf("Invalid number of philosophers.\n");
        return;
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            printf("P %d and P %d are granted to eat\n", hungry[i], hungry[j]);
        }
    }
}
```

```

for (int k = 0; k < n; k++) {
    if (k != i && k != j) {
        printf("P %d is waiting\n", hungry[k]);
    }
}
}

int main() {
    int total_philosophers, hungry_count;
    int hungry_positions[MAX_PHILOSOPHERS];
    printf("DINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: ");
    scanf("%d", &total_philosophers);
    if (total_philosophers > MAX_PHILOSOPHERS || total_philosophers < 2) {
        printf("Invalid number of philosophers.\n");
        return 1;
    }
    printf("How many are hungry: ");
    scanf("%d", &hungry_count);
    if (hungry_count < 1 || hungry_count > total_philosophers) {
        printf("Invalid number of hungry philosophers.\n");
        return 1;
    }
    for (int i = 0; i < hungry_count; i++) {
        printf("Enter philosopher %d position: ", i + 1);
        scanf("%d", &hungry_positions[i]);
        if (hungry_positions[i] < 0 || hungry_positions[i] >= total_philosophers) {
            printf("Invalid philosopher position.\n");
            return 1;
        }
    }
}

```

```
}

int choice;

while (1) {

    printf("\n1. One can eat at a time\n");
    printf("2. Two can eat at a time\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {

        case 1:
            allow_one_to_eat(hungry_positions, hungry_count);
            break;

        case 2:
            allow_two_to_eat(hungry_positions, hungry_count);
            break;

        case 3:
            exit(0);

        default:
            printf("Invalid choice\n");
    }
}

return 0;
}
```

Output:

```
DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
How many are hungry: 2
Enter philosopher 1 position: 1
Enter philosopher 2 position: 4

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
P 1 is granted to eat
P 4 is waiting
P 4 is granted to eat

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
P 1 and P 4 are granted to eat

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: -
```

7. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>

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

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

    int maximum[n][m];
    printf("Enter the maximum resources for each process:\n");
    for (int i = 0; i < n; i++) {
```

```

for (int j = 0; j < m; j++) {
    scanf("%d", &maximum[i][j]);
}

}

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

int need[n][m];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        need[i][j] = maximum[i][j] - allocation[i][j];
    }
}

printf(" Process Allocation Max Need      \n");
for (int i = 0; i < n; i++) {
    printf(" | P%d | ", i + 1);
    for (int j = 0; j < m; j++) {
        printf("%d ", allocation[i][j]);
    }
    printf(" | ");
    for (int j = 0; j < m; j++) {
        printf("%d ", maximum[i][j]);
    }
    printf(" | ");
    for (int j = 0; j < m; j++) {
        printf("%d ", need[i][j]);
    }
}

```

```

printf("|\n");
}

int work[m];
for (int i = 0; i < m; i++) {
    work[i] = available[i];
}

int finish[n];
for (int i = 0; i < n; i++) {
    finish[i] = 0;
}

int safeSequence[n];
int count = 0;
int safe = 1;
while (count < n) {
    int found = 0;
    for (int i = 0; i < n; i++) {
        if (finish[i] == 0) {
            int j;
            for (j = 0; j < m; j++) {
                if (need[i][j] > work[j]) {
                    break;
                }
            }
            if (j == m) {
                for (j = 0; j < m; j++) {
                    work[j] += allocation[i][j];
                }
                finish[i] = 1;
                safeSequence[count++] = i;
                found = 1;
            }
        }
    }
}

```

```
    }
}

if (!found) {
    safe = 0;
    break;
}

if (safe) {
    printf("The system is in a safe state.\n");
    printf("Safety sequence: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", safeSequence[i] + 1);
    }
    printf("\n");
} else {
    printf("The system is in an unsafe state and might lead to deadlock.\n");
}
return 0;
}
```

Output:

```

Enter the number of processes: 5
Enter the number of resources: 3
Enter the available resources: 3 3 2
Enter the maximum resources for each process:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocated resources for each process:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2
Process   Allocation   Max   Need
| P1      | 0 1 0 | 7 5 3 | 7 4 3 |
| P2      | 3 0 2 | 3 2 2 | 0 2 0 |
| P3      | 3 0 2 | 9 0 2 | 6 0 0 |
| P4      | 2 1 1 | 2 2 2 | 0 1 1 |
| P5      | 0 0 2 | 4 3 3 | 4 3 1 |
The system is in a safe state.
Safety sequence: P2 P3 P4 P5 P1

Process returned 0 (0x0)  execution time : 62.329 s
Press any key to continue.

```

8. Write a C program to simulate deadlock detection

```

#include<stdio.h>

void main(){
    int n,m,i,j;
    printf("Enter the number of processes and number of types of resources:\n");
    scanf("%d %d",&n,&m);
    int max[n][m],need[n][m],all[n][m],ava[m],flag=1,finish[n],dead[n],c=0;
    printf("Enter the maximum number of each type of resource needed by each process:\n");
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            scanf("%d",&max[i][j]);
        }
    }
    printf("Enter the allocated number of each type of resource needed by each process:\n");
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){

```

```

scanf("%d",&all[i][j]);
}

}

printf("Enter the available number of each type of resource:\n");
for(j=0;j<m;j++){
    scanf("%d",&ava[j]);
}

for(i=0;i<n;i++){
    for(j=0;j<m;j++)
    {
        need[i][j]=max[i][j]-all[i][j];
    }
}

for(i=0;i<n;i++){
    finish[i]=0;
}

while(flag){

    flag=0;
    for(i=0;i<n;i++){
        c=0;
        for(j=0;j<m;j++){
            if(finish[i]==0 && need[i][j]<=ava[j]){
                c++;
                if(c==m){
                    for(j=0;j<m;j++){
                        ava[j]+=all[i][j];
                    }
                    finish[i]=1;
                    flag=1;
                }
            }
            if(finish[i]==1){
                i=n;
            }
        }
    }
}

```

```
        }
    }
}
}

j=0;
flag=0;
for(i=0;i<n;i++){
    if(finish[i]==0){
        dead[j]=i;
        j++;
        flag=1;
    }
}
if(flag==1){
    printf("Deadlock has occurred:\n");
    printf("The deadlock processes are:\n");
    for(i=0;i<n;i++){
        printf("P%d ",dead[i]);
    }
}
else
printf("No deadlock has occurred!\n");
}
```

Output:

```
Enter the number of processes and number of types of resources:  
4 3  
Enter the allocated number of each type of resource needed by each process:  
1 0 2  
2 1 1  
1 0 3  
2 2 2  
Enter the available number of each type of resource:  
0 0 0  
Enter the request number of each type of resource needed by each process:  
0 0 1  
1 0 2  
0 0 0  
3 3 0  
Deadlock has occurred:  
The deadlock processes are:  
P3  
Process returned 1 (0x1) execution time : 47.994 s  
Press any key to continue.  
-
```

9. Write a C program to simulate the following contiguous memory allocation

techniques

a) Worst-fit

b) Best-fit

c) First-fit

```
#include <stdio.h>

struct Block {
    int block_no;
    int block_size;
    int is_free;
};

struct File {
    int file_no;
    int file_size;
};

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("Memory Management Scheme - First Fit\n");
    printf("File_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for (int i = 0; i < n_files; i++) {
```

```

for (int j = 0; j < n_blocks; j++) {

    if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {

        blocks[j].is_free = 0;

        printf("%d\t%d\t%d\t%d\t%d\n", files[i].file_no, files[i].file_size, blocks[j].block_no,
blocks[j].block_size, blocks[j].block_size - files[i].file_size);

        break;
    }
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {

    printf("Memory Management Scheme - Worst Fit\n");

    printf("File_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");

    for (int i = 0; i < n_files; i++) {

        int worst_fit_block = -1;

        int max_fragment = -1;

        for (int j = 0; j < n_blocks; j++) {

            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {

                int fragment = blocks[j].block_size - files[i].file_size;

                if (fragment > max_fragment) {

                    max_fragment = fragment;

                    worst_fit_block = j;
                }
            }
        }

        if (worst_fit_block != -1) {

            blocks[worst_fit_block].is_free = 0;

            printf("%d\t%d\t%d\t%d\t%d\t%d\n", files[i].file_no, files[i].file_size,
blocks[worst_fit_block].block_no, blocks[worst_fit_block].block_size, max_fragment);
        }
    }
}

```

```

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("Memory Management Scheme - Best Fit\n");
    printf("File_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for (int i = 0; i < n_files; i++) {
        int best_fit_block = -1;
        int min_fragment = 10000;
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                if (fragment < min_fragment) {
                    min_fragment = fragment;
                    best_fit_block = j;
                }
            }
        }
        if (best_fit_block != -1) {
            blocks[best_fit_block].is_free = 0;
            printf("%d\t%d\t%d\t%d\t%d\n", files[i].file_no, files[i].file_size,
                   blocks[best_fit_block].block_no, blocks[best_fit_block].block_size, min_fragment);
        }
    }
}

int main() {
    int n_blocks, n_files;
    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);
    printf("Enter the number of files: ");
    scanf("%d", &n_files);
    struct Block blocks[n_blocks];
    for (int i = 0; i < n_blocks; i++) {
        blocks[i].block_no = i + 1;
        printf("Enter the size of block %d: ", i + 1);
    }
}

```

```

scanf("%d", &blocks[i].block_size);

blocks[i].is_free = 1;

}

struct File files[n_files];

for (int i = 0; i < n_files; i++) {

    files[i].file_no = i + 1;

    printf("Enter the size of file %d: ", i + 1);

    scanf("%d", &files[i].file_size);

}

firstFit(blocks, n_blocks, files, n_files);

printf("\n");

for (int i = 0; i < n_blocks; i++) {

    blocks[i].is_free = 1;

}

worstFit(blocks, n_blocks, files, n_files);

printf("\n");

for (int i = 0; i < n_blocks; i++) {

    blocks[i].is_free = 1;

}

bestFit(blocks, n_blocks, files, n_files);

return 0;
}

```

Output:

```

Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of block 1: 5
Enter the size of block 2: 2
Enter the size of block 3: 7
Enter the size of file 1: 1
Enter the size of file 2: 4
Memory Management Scheme - First Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1                1                5                4
2            4                3                7                3

Memory Management Scheme - Worst Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1                3                7                6
2            4                1                5                1

Memory Management Scheme - Best Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1                2                2                1
2            4                1                5                1

Process returned 0 (0x0)   execution time : 38.325 s
Press any key to continue.
-
```

10. write a c program to stimulate page replacement algorithms

a)FIFO

b)LRU

c)Optimal

```

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

void print_frames(int frame[], int capacity, int page_faults) {
    for (int i = 0; i < capacity; i++) {
        if (frame[i] == -1)
            printf("- ");
        else
            printf("%d ", frame[i]);
    }
    if (page_faults > 0)
        printf("PF No. %d", page_faults);
    printf("\n");
}

void fifo(int pages[], int n, int capacity) {
    int frame[capacity], index = 0, page_faults = 0;
    for (int i = 0; i < capacity; i++)

```

```

frame[i] = -1;

printf("FIFO Page Replacement Process:\n");
for (int i = 0; i < n; i++) {
    int found = 0;
    for (int j = 0; j < capacity; j++) {
        if (frame[j] == pages[i]) {
            found = 1;
            break;
        }
    }
    if (!found) {
        frame[index] = pages[i];
        index = (index + 1) % capacity;
        page_faults++;
    }
    print_frames(frame, capacity, found ? 0 : page_faults);
}
printf("Total Page Faults using FIFO: %d\n\n", page_faults);

void lru(int pages[], int n, int capacity) {
    int frame[capacity], counter[capacity], time = 0, page_faults = 0;
    for (int i = 0; i < capacity; i++) {
        frame[i] = -1;
        counter[i] = 0;
    }

    printf("LRU Page Replacement Process:\n");
    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < capacity; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                counter[j] = time++;
                break;
            }
        }
        if (!found) {
            int min = INT_MAX, min_index = -1;
            for (int j = 0; j < capacity; j++) {
                if (counter[j] < min) {
                    min = counter[j];
                    min_index = j;
                }
            }
            frame[min_index] = pages[i];
            counter[min_index] = time++;
            page_faults++;
        }
        print_frames(frame, capacity, found ? 0 : page_faults);
    }
}

```

```

    }
    printf("Total Page Faults using LRU: %d\n\n", page_faults);
}

void optimal(int pages[], int n, int capacity) {
    int frame[capacity], page_faults = 0;
    for (int i = 0; i < capacity; i++)
        frame[i] = -1;

    printf("Optimal Page Replacement Process:\n");
    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < capacity; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            int farthest = i + 1, index = -1;
            for (int j = 0; j < capacity; j++) {
                int k;
                for (k = i + 1; k < n; k++) {
                    if (frame[j] == pages[k])
                        break;
                }
                if (k > farthest) {
                    farthest = k;
                    index = j;
                }
            }
            if (index == -1) {
                for (int j = 0; j < capacity; j++) {
                    if (frame[j] == -1) {
                        index = j;
                        break;
                    }
                }
            }
            frame[index] = pages[i];
            page_faults++;
        }
        print_frames(frame, capacity, found ? 0 : page_faults);
    }
    printf("Total Page Faults using Optimal: %d\n\n", page_faults);
}

int main() {
    int n, capacity;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
}

```

```

int *pages = (int*)malloc(n * sizeof(int));
printf("Enter the pages: ");
for (int i = 0; i < n; i++)
    scanf("%d", &pages[i]);
printf("Enter the frame capacity: ");
scanf("%d", &capacity);

printf("\nPages: ");
for (int i = 0; i < n; i++)
    printf("%d ", pages[i]);
printf("\n\n");

fifo(pages, n, capacity);
lru(pages, n, capacity);
optimal(pages, n, capacity);

free(pages);
return 0;
}

```

Output :

```

2 3 0 PF No. 6
4 3 0 PF No. 7
4 2 0 PF No. 8
4 2 3 PF No. 9
0 2 3 PF No. 10
0 2 3
0 2 3
0 1 3 PF No. 11
0 1 2 PF No. 12
0 1 2
0 1 2
7 1 2 PF No. 13
7 0 2 PF No. 14
7 0 1 PF No. 15
Total Page Faults using FIFO: 15

LRU Page Replacement Process:
7 -- PF No. 1
0 -- PF No. 2
0 1 - PF No. 3
0 1 2 PF No. 4
0 1 2
0 3 2 PF No. 5
0 3 2
0 3 4 PF No. 6
0 2 4 PF No. 7
3 2 4 PF No. 8
3 2 0 PF No. 9
3 2 0
3 2 0
3 2 1 PF No. 10
3 2 1
0 2 1 PF No. 11

```

```
C:\Users\saisr\OneDrive\Desktop + X - O X  
0 2 1 PF No. 11  
0 2 1  
0 7 1 PF No. 12  
0 7 1  
0 7 1  
Total Page Faults using LRU: 12  
  
Optimal Page Replacement Process:  
7 - - PF No. 1  
7 0 - PF No. 2  
7 0 1 PF No. 3  
2 0 1 PF No. 4  
2 0 1  
2 0 3 PF No. 5  
2 0 3  
2 4 3 PF No. 6  
2 4 3  
2 4 3  
2 0 3 PF No. 7  
2 0 3  
2 0 3  
2 0 1 PF No. 8  
2 0 1  
2 0 1  
7 0 1 PF No. 9  
7 0 1  
7 0 1  
Total Page Faults using Optimal: 9  
  
Process returned 0 (0x0) execution time : 73.003 s  
Press any key to continue.
```

```
C:\Users\saisr\OneDrive\Desktop + X - O X  
Enter the number of pages: 20  
Enter the pages: 7  
0  
1  
2  
0  
3  
0  
4  
2  
3  
0  
1  
2  
0  
1  
7  
0  
1  
Enter the frame capacity: 3  
  
Pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  
  
FIFO Page Replacement Process:  
7 - - PF No. 1  
7 0 - PF No. 2  
7 0 1 PF No. 3  
2 0 1 PF No. 4  
2 0 1  
2 3 1 PF No. 5  
2 3 0 PF No. 6
```

INDEX

Name : Class :

Section : Roll No. : Subject :

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
		LAB 1		
1	8/15/24	a) FCFS b) SJF Non-preemptive c) Preemptive		7
2	15/15/24	a) Priority ^{LAB-2} b) Round Robin		
3	22/15/24	Multilevel Queue		
4	8/15/24	a) Rate Monotonic Scheduling b) Earliest Deadline First scheduling c) Proportionality scheduling.		
5	12/16/24	producer-consumer problem using semaphores.		
6	19/16/24	a) Banker's algorithm for the purpose of deadlock avoidance b) Dining philosopher.		
7	3/17/24	Deadlock detection		
8	3/17/24	Memory allocation a) Worst-fit b) Best fit c) T-Best-fit		8/10/24 Comp
9	10/16/24	Page Replacement Algorithms 1) FIFO 2) LRU 3) Optimal.		

15/24

Date _____
Page _____

LAB-1

Scheduling programming Algorithm:

- " a) write a c program to stimulate the following non-pre-emptive CPU-scheduling algorithm to find turnaround time and waiting time.

#include <stdio.h>

#define Max-process 10;
void fcfs(int n, int at[], int bt[]);

{

int ct[Max-process];

int tat[Max-process];

int wt[Max-process];

int total_wt=0;

int total_tat=0;

int current_time=0;

for(i=0;i<n;i++)

{

ct[i]=-1;

}

for (int i=0; i<n; i++)

{

if (current_time<at[i])

{

current_time=at[i];

3

ct[i]=current_time+bt[i];

current_time=ct[i];

3

for (int i=0 ; i<n ; i++)

{

tat[i]=ct[i]-at[i];

total_tat+=tat[i];

T-7A:

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

printf("In process at Arrival time %d Burst time %d completion time %d Turnaround time %d waiting time %d\n");
for (i=0; i<n; i++) {
    printf("%d %d %d %d %d %d %d\n",
        i+1, at[i], bt[i], ctf[i], tat[i], wt[i]);
}

printf("In Average waiting time: %.2f", (float)totalwt/n);
printf("In Average turnaround time: %.2f", (float)totaltat/n);

}

int main() {
    int n, i;
    int at[10], bt[10];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[10], bt[10];
    printf("Enter the arrival time: \n");
    for (i=0; i<n; i++) {
        printf("Enter the burst time: \n");
        scanf("%d", &bt[i]);
    }

    printf("Enter the burst time: \n");
    for (i=0; i<n; i++) {
        scanf("%d", &bt[i]);
    }
    return 0;
}
```

Output:

Enter the number of processes: 4

Enter the arrival time:

0

1

5

6

Enter the burst time: 1, 2, 3, 4

2

2

3

4

process Arrival time Burst time completion time Turnaround time waiting time

P ₁	0	1	2	3	2	0
----------------	---	---	---	---	---	---

P ₂	1	2	3	4	3	1
----------------	---	---	---	---	---	---

P ₃	5	3	8	9	3	6
----------------	---	---	---	---	---	---

P ₄	6	4	10	12	6	2
----------------	---	---	----	----	---	---

Average waiting time: 0.75 min.

Average turnaround time: 3.50

First come first serve

Shortest job first

Priority scheduling

Round robin scheduling

Preemptive scheduling

5/5/24

b) BJT Non-preemptive

#include <stdio.h>

int main()

{

int n, i, j, temp1, temp2;

float atat = 0, awt = 0;

printf("Enter the number of processes: ");

scanf("%d", &n);

int atime[n], btime[n], clime[n], tatime[n], wtimes[n], pid[n];

printf("Enter process IDs: ");

for (i=0; i<n; i++)

{

scanf("%d", &pid[i]);

printf("Enter the arrival times: ");

for (i=0; i<n; i++)

{

scanf("%d", &atime[i]);

}

printf("Enter burst times: ");

for (i=0; i<n; i++)

{

scanf("%d", &btime[i]);

}

for (i=0; i<n-1; i++)

{

for (j=i+1; j<n; j++)

{

if (btime[i] < btime[j])

{

temp1 = btime[i];

$btime2[j] = btime2[j+1];$

$btime2[j+1] = temp1;$

$temp2 = pid[j];$

$pid[j] = pid[j+1];$

$pid[j+1] = temp2;$

}

}

for ($i=0; i < n; i++$)

{

if ($i == 0$)

{

$ctime3[i] = atime1[i] + btime2[i];$

}

else

{

if ($ctime3[i-1] < atime1[i]$)

$ctime3[i] = atime1[i] - ctime3[i-1] + 3 * ctime3[i-1] +$

$btime2[i];$

}

else

$ctime3[i] = ctime3[i-1] + btime2[i];$

}

}

for ($i=0; i < n; i++$)

{

$tattime4[i] = ctime3[i] * atime1[i];$

}

```

for (i=0; i<n; i++) {
    wtime5[i] = tat - time4[i] - btime2[i];
}

for (i=0; i<n; i++) {
    atat = atat + b1 - time4[i];
}

atot = (atat / n);

for (i=0; i<n; i++) {
    awt += awt + wtime5[i];
}

printf("Process ID, Arrival Time, TAT, Burst time,\n"
       "Complete Time, Turn Around Time, Waiting Time\n"
       "%d,%d,%d,%d,%d,%d,%d,%d,%d\n",
       pid[i], afime1[i], btime1[i], time3[i], tat[i],
       wtime5[i]);
}

printf("Average Turn Around Time: %f\n", atat);
printf("Average waiting Time: %f\n", awt);
return 0;
}

```

Output:

Process	Arrival time	Burst time	Completion Time	Turnaround time	Waiting time
P ₁	0	6	6	6	3
P ₂	0	8	14	8	16
P ₃	0	7	21	14	9
P ₄	0	3	3	3	0
P ₅			27	24	
					Turnaround time = 24
					Average of Turnaround time = 13
					Average of waiting time = 7.2

c) Pre-emptive

```
#include <stdio.h>
#define Max 10
int find_min(int arr[], int n)
{
    int min = arr[0];
    int index = 0;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] < min)
            if (arr[i] < min)
                min = arr[i];
                index = i;
    }
    return index;
}
```

void SJF-preemptive (int n, int at[], int bt[]):

```
int ct[MaxJ]={0};  
int tat[MaxJ]=0;  
int wt[MaxJ]=0;  
int rt[MaxJ];  
int total_wt=0;  
int total_tat=0;  
for (int i=0; i<n; i++)  
{  
    rt[i]=bt[i];  
}  
int current_time=0;  
int completed_processes=0;  
while (completed_processes < n) {  
    int available_processes[Max];  
    int available_count=0;  
    for (int i=0; i<n; i++)  
    {  
        if (rt[i] <= current_time && rt[i] > 0);  
            available_processes[available_count]=i;  
            available_count++;  
    }  
    int shortest_job_index=available_processes  
    [find_min(1, available_count)];  
    rt[shortest_job_index]--;  
    current_time++;
```

```

if (rt[shortest-job-index] == 0)
    current_time = time + burst_time[shortest-job-index];
else
    completed_processes += 1;
    current_time = time + burst_time[shortest-job-index];
    shortest-job-index = current_time - burst_time[shortest-job-index];
    if (shortest-job-index < 0)
        shortest-job-index = 0;

```

```

a) if (shortest-job-index) != shortest-job-index - 1
    b) if (shortest-job-index) == shortest-job-index - 1

```

```

total = total + burst_time[shortest-job-index];
total -= 1;

```

```

total -= 1;

```

```

} // end of while loop
}
```

```

printf("In process %d Arrival Time %d Burst Time %d Completion Time %d Turn around time %d Waiting time %d\n",
    for (int i=0; i<n; i++)

```

```

    printf("%d %d %d %d %d %d %d\n",
        arr[i], btr[i], crr[i], stat[i], wt[i]);
}

```

```

printf("Average waiting time : %f", float)
    total_wt/n);

```

```

printf("Average turnaround time : %f", float)
    total_bt/n);
}

```

```

int main()
{
    int n;
    int arr[Max];
    int btr[Max];
    int crr[Max];
    int stat[Max];
    int wt[Max];
    float avgwt, avgtat;
}
```

```

    int i;
}
```

```

    printf("Enter the number of processes : ");

```

```

    scanf("%d", &n);

```

```

    int ar[Max];
    int bt[Max];

```

```

    printf("Enter the arrival time : ");

```

```

scanf("%d", &n);
int at[max], bt[max];
printf("Enter the arrival time: \n");
for (int i=0; i<n; i++) {
    if (at[i] == -1) {
        scanf("%d", &at[i]);
    }
}
printf("Enter the burst time: \n");
for (int i=0; i<n; i++) {
    if (bt[i] == -1) {
        scanf("%d", &bt[i]);
    }
}

```

~~act 30~~

process id : arrival time burst time CTAT WT

return 0; // main function

3

process id : arrival time burst time CTAT WT

P1 2 1 3 8 0

P2 4 1 5 7 6 1

P3 4 1 8 18 3

P4 1 0 6 13 7

P5 2 3 16 31 11

Average Waiting time = 4.40

Average turnaround time = 7.60

S-8A1

$$at[0] = at[0] + bt[0];$$

$$tat[0] = ct[0] - at[0];$$

$$wt[0] = tat[0] - bt - copy[0];$$

$$totbt = wt[0];$$

$$total - totbt = tat[i];$$

}

printf("In Process %d Arrival Time %d burst time %d

Completion time %d Turnaround time %d waiting time %d\n", i+1, at[i], bt[i], copy[i], tat[i], ct[i], wt[i]);

for (int i=0; i<n; i++)

{

printf("Process %d Arrival time %d burst time %d completion time %d turnaround time %d waiting time %d\n", i+1, at[i], bt[i], copy[i], tat[i], ct[i], wt[i]);

}

printf("Average waiting time %f", float)

(float)totwt / totbt);

printf("Average turnaround time %f", float)

(float)totaltat / totbt);

)

int main()

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

int at[MAX], bt[MAX], pt[MAX];

printf("Enter the arrival time: ");

scanf("%d", &at[0]);

int at1[MAX], bt1[MAX], pt1[MAX];

printf("Enter the arrival time: ");

for (int i=0; i<n; i++)

scanf("%d", &at1[i]));

printfill(i) {
 fill(i);
 cout << "The burst times are : ";

 for (int i = 0; i < n; i++)
 cout << bt[i] << " ";
 cout << endl;

priority - non-preemptive Round Robin

return 0;

}

Output:

PID	Priority	AT	BT	CT	TAT	WT	RT
P ₁	10	0	5	12	12	7	6
P ₂	20	1	14	25	25	24	0
P ₃	30	2	2	4	2	0	0
P ₄	40	4	0	5	5	0	0

Average TAT = 15.5

Average WT = 2.5

15/12/24

b) Round ROBIN scheduling:

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

```
#define MAX 10
```

```
#define quantum 2
```

```
void round_robin (int n, int tq, int bt[])
```

```
{
```

```
    int ct [MAX] = {0};
```

```
    int tat [MAX] = {0};
```

```
    int wtf [MAX] = {0};
```

```
    int total_wt = 0;
```

```
    int total_tat = 0;
```

```
    int bt_copy [MAX];
```

```
    for (int i=0; i<n; i++)
```

```
        bt_copy[i] = bt[i];
```

```
}
```

```
    int time = 0;
```

```
    int i = 0;
```

```
    while (1)
```

```
{
```

```
    int done = 1;
```

```
    for (int j=0; j<n; j++)
```

```
{
```

```
        if (bt_copy[j] > 0)
```

```
{
```

```
            done = 0;
```

```
            if (bt_copy[j] > quantum)
```

```
{
```

```
                time += quantum;
```

```
}
```

```
else
```

```
{ time += bt_copy[j];
```



b

for (int i=0; i<n; i++) {
 bt[i] = copy[i];
 ct[i] = time[i];
 tat[i] = ct[i] - at[i];
 wt[i] = tat[i] - bt[i];
 total_tat += tat[i];
 total_wt += wt[i];
}

}

{ Done } will occur when all processes
have ;

3 } waiting time.

printf("In process %d arrival time %d burst time %d
completion time %d turn around time %d waiting
time in %d",

for (int i=0; i<n; i++)

{

printf("%d %d %d %d %d %d\n",

i+1, at[i], bt[i], ct[i], tat[i], wt[i]);

}

printf("Average waiting time : %f\n", (float) total_wt/n);

printf("Average turn around time : %f\n", (float) total_tat/n);

int main()

int n;

printf("Enter the number of process:");

scanf("%d", &n);

int at[100], bt[100], t[100];

printf("Enter arrival time: ");

for (int i=0; i<n; i++)

{

scanf("%d", &at[i]);

printf("Enter burst time: ");

for (int i=0; i<n; i++)

{

scanf("%d", &bt[i]);

}

cout <<

"Enter the number of process: "

Enter arrival time

0

1

2

3

4

Enter the burst time:

3

1

EX-18-9A

Process Flowchart

2

Activities that will be done

3 Short description of activities will be

PROCESS AT BT CT TAT WT

1	0	5	14	14	3	9
---	---	---	----	----	---	---

2	1	3	12	11	2	8
---	---	---	----	----	---	---

3	2	1	15	14	2	2
---	---	---	----	----	---	---

4	3	2	7	4	2	2
---	---	---	---	---	---	---

5	4	3	13	9	6	6
---	---	---	----	---	---	---

Waiting time

Average WT : 5.40

Average TAT : 8.20

Process flowchart of the activities and their sequence
1. Cleaned tank & Piping system

2. Filled water in tank

3. Pumping water

4

5

Water analysis to check the quality of water

independent task, 10 min

6. 10 min

Water analysis to check the quality of water

independent task, 10 min

7. 10 min

8. 10 min

9. 10 min

10. Independent task, 10 min

11. 10 min

12. 10 min

5/6/24

LAB-3 Program.

Multilevel queue //

```
#include < stdio.h >
void findWaitingTime (int processes[], int n, int bt[], int at[],
int wt[])
{
    wt[0] = 0;
    for (int i = 1, j < n; i++)
    {
        wt[i] = bt[i - 1] + wt[i - 1] - at[i - 1];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}
```

```
void findTurnaroundTime (int processes[], int n, int bt[],
int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
    }
}
```

```
void RoundRobin (int processes[], int n, int bt[], int at[],
int quantum)
{
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    int remaining_bt[n];
    int completed = 0;
    int time = 0;
    for (int i = 0; i < n; i++)
    {
        remaining_bt[i] = bt[i];
    }
}
```

```

while (completed < n)
{
    for (int i = 0; i < n; i++)
    {
        if (remaining_bt[i] > 0 &amp; at[i] <= time)
        {
            if (remaining_bt[i] <= quantum)
                time += remaining_bt[i];
            else
            {
                time += quantum;
                completed++;
            }
        }
        else
    }
}

```

if (time <= quantum)

remaining_bt[i] = quantum; time += quantum;

y

y

z

else if (time > quantum)

find waiting Time (processes, n, bt, at, wt);

find Turnaround Time (processes, n, bt, wt, tat);

printf ("processes burst time arrival time waiting time

Turnaround time completion time\n");

for (int i = 0; i < n; i++)

{

```

        printf ("p.%d\tbt.%d\tat.%d\twt.%d\ttat.%d\n") processes[i]
            .at[i], processes[i].bt[i], processes[i].wt[i],
            processes[i].tat[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }
}

```

```

printf("Average waiting Time (Round Robin) = %f\n",
      (float) total_wt / n);
printf("Average Turnaround time (Round Robin) = %f\n",
      (float) total_tat / n);
}

void fcfs_sltm(processes[], int n, int bt[], int at[])
{
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, at, wt);
    findTurnaroundTime(processes, n, bt, wt, tat);
    printf("processes Burst Time Arrival Time waiting time
           turnaround time completion time\n");
    for (int i=0; i<n; i++)
    {
        ct[i] = at[i] + bt[i];
        printf("%d %d %d %d %d %d %d\n",
               process[i], bt[i], at[i], wt[i], tat[i], ct[i]);
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf("Average waiting Time (FCFS) = %f\n", (float)
          total_wt / n);
    printf("Average Turnaround Time (FCFS) = %f\n", (float)
          total_tat / n);
}

```

int main()

{

int processes[] = {1, 2, 3, 4, 5};

int n = size_of(processes) / size_of(processes[0]);

int bt = {10, 5, 8, 12, 15};

int at[] = {0, 1, 2, 3, 4};

int quantum = 2;

roundRobin(processes, n, bt, at, quantum);

FCFS(processes, n, bt);

return 0;

}

Output

PROCESS	BT	AT	WT	TAT	CT
P1	10	0	0	10	39
P2	5	1	10	15	23
P3	8	2	14	22	33
P4	12	3	20	32	45
P5	15	4	29	44	50

$$\text{Avg WT (Round Robin)} = \frac{10+60}{5} = 14.60$$

$$\text{Avg Turnaround time (Round Robin)} = \frac{39+23+33+45+50}{5} = 36.60$$

process	BT	AT	WT	TAT	CT
P1	10	0	0	10	10
P2	5	1	10	15	15
P3	8	2	14	22	22
P4	12	3	20	32	32
P5	15	4	29	44	44

$$\text{Avg WT (FCFS)} = \frac{10+60}{5} = 14.60$$

$$\text{Avg TAT (FCFS)} = \frac{39+23+33+45+50}{5} = 36.60$$

5/6/24

QAPP program.

a) Rate Monotonic //

#include <stdio.h>

Struct process {

int execution-time;

int time-period;

};

int lcm(int a, int b);

{

int max = (a > b) ? a : b;

while(1)

{

if (max - a == 0 && max - b == 0)

return max;

max + 1;

}

}

int is-schedulable (struct process processes[], int n)

float utilization = 0.0;

for (int i = 0; i < n; i++)

{

utilization += (float) processes[i].execution-time /

processes[i].time-period;

}

}

return utilization < 1.0;

}

int main()

{

Struct process processes[5];

{

{ 3, 20 },

{ 2, 15 },

{ 2, 10 },

}

int n = sizeof(processes) / sizeof(processes[0]);

if (!is_schedulable(processes, n))

{

printf("The given set of processes is not
schedule-able");

return 0;

}

int scheduling_time = lcm(processes[0].time-period,
processes[1].time-period);

Scheduling-time = lcm(scheduling-time, processes[2].
time-period);

printf("Execution order: 1n")

if (++processes[1].time-period == 0)

printf("P2");

if (++processes[2].time-period == 0)

printf("P3");

if (++processes[0].time-period == 0)

printf("P1");

{

printf("1n");

return 0;

};

Output:

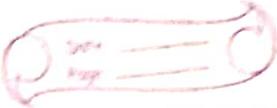
Execution order: P1 P2 P3 P1 P2

P2 P3 P1 P2 P3 P1

5/6/24

b) Earliest ready time first scheduling // min

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void Sort (int proc[], int d[], int b[], int pt[], int n)
{
    int item p = 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[i] = temp;
            }
        }
    }
}
```



int gcd (int a, int b)

```
{  
    int r; // remainder after division  
    while (b > 0)  
    {  
        r = a % b; // calculate remainder  
        a = b; // update a to b  
        b = r; // update b to remainder  
    }  
    return a;  
}
```

int

```
lcmul (int p[], int n)  
{  
    int lcm = p[0]; // initial value  
    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], r[n]; // memory

printf ("Enter the CPU burst times: ");

for (int i = 0; i < n; i++)

```
{  
    scanf ("%d", &proc[i]); // input r[i]
```

```

    rem[i] = b[i];
    printf("Enter the deadline: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &df[i]);
    }
    printf("Enter the time periods: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &pt[i]);
    }
    proc[i] = i + 1;
    sort(proc, df, pt, n);
    int l = lcm(u1(pt, n));
    printf("In Earliest Deadline Scheduling: \n");
    printf("Process Burst & deadline if period \n");
    for (int i = 0; i < n; i++) {
        printf("%d %d %d %d \n", proc[i], b[i],
               df[i], pt[i]);
    }
    printf("Scheduling occurs for %d ms \n", l);
    int time = 0, prev = 0, x = 0;
    int nextDeadline[n];
    for (int i = 0; i < n; i++) {
        nextDeadline[i] = df[i];
    }
    while (time < l) {
        for (int i = 0; i < n; i++) {
            if (time == nextDeadline[i]) {
                rem[i] = b[i];
                time += pt[i];
            }
        }
    }
}

```



```
if (time + pt[i] == 0 && time != 0) {
    rem[i] = b[i];
}
else if (time + pt[i] >= 0 && time != 0) {
    nextDeadline[i] = time + td[i];
    rem[i] = b[i];
}
}

int minDeadline = 1;
int taskToExecute = -1;
for (int i = 0; i < n; i++) {
    if (rem[i] > 0 && nextDeadline[i] < minDeadline) {
        minDeadline = nextDeadline[i];
        taskToExecute = i;
    }
}

if (taskToExecute != -1) {
    printf("At %d ms: Task %d is running.\n", time, taskToExecute);
    rem[taskToExecute] = -minDeadline;
}
else {
    printf("At %d ms: CPU is idle.\n", time);
}

time++;
}
```

Output

Enter the number of process : 3

Enter the CPU burst times:

3

2

2

Enter the deadlines:

三

4

8

for the time periods:

20. *Leptospirosis* (Lepto) - *Leptospiral* (Lepto) - *Lepto*

5

16

earliest deadline scheduling.

PIO	Burst	deadline	period
2	2	4	5
3	3	7	20
1	2	8	10

Scheduling occurs for ~~some~~

Ans: Task 2 is running

1ms : Task 2

2ms : Task 1

3ms : Task,

urms : TASHI

5ms : TOSIC2

6ms, Task 2

7 ms : Task 3

: TASK 2

8th: Task 2
Ans:

~~q ms~~, CPU is

DMS : TDSR

11mS = T98 K

c) proportionality scheduling

```

#include <stdio.h>
#include <stdlib.h> // for rand()
#include <time.h> // for time()
#define MAX_TASKS 10
#define MAX_TICKETS 100
#define TIME_UNIT_DURATION_MS 100
#define TICKET_BONUS 50

Struct Task {
    int tid;
    int tickets;
};

void schedule (Struct Task tasks[], int num_tasks,
    int time_span_ms) {
    int total_tickets = 0;
    for (int i=0; i<num_tasks; i++) {
        tasks[i].tickets = rand() % 100;
        total_tickets += tasks[i].tickets;
    }
    srand (time(NULL));
    int current_time = 0;
    int completed_tasks = 0;
    printf ("Process scheduling...\n");
    while (completed_tasks < num_tasks) {
        int winning_ticket = rand() % total_tickets;
        int cumulative_tickets = 0;
        for (int i=0; i<num_tasks; i++) {
            cumulative_tickets += tasks[i].tickets;
            if (winning_ticket < cumulative_tickets) {
                tasks[i].tickets = 0;
                completed_tasks++;
                break;
            }
        }
    }
}

```

```

int cumulative_ticks = 0;
for (int i=0; i<num_tasks; i++) {
    cumulative_ticks += tasks[i].ticks;
    if (running_ticks < cumulative_ticks) {
        printf("Time %d ms: Task %d is running in",
               current_time, current_time + tasks[i].id),
        current_time);
        break;
    }
}

```

(Completed-tasks++;

*time-span-ms = current-time * TIME-UNIT-duration-ms;

int main()

{

struct Task {Max-tasks};

int num_tasks;

int num_tasks;

int time-span-ms;

printf("Enter the number of tasks: ");

scanf("%d", &num_tasks);

If (num_tasks <= 0 || num_tasks > Max-tasks)

{

printf("Invalid number of tasks, please enter a number between 1 and %d\n", Max-tasks);

}

```
print("Enter number of tickets for each task:\n");
for (int i=0; i< num-tasks; i++)
{
    tasks[i].tid = i+1;
    print("Task " + i + " tickets: ", tasks[i].tid);
    scanf("%d", &tasks[i].tickets);
}

printf("In running tasks:\n");
Schedule(tasks, num-tasks, time-expansions);
printf("In time span of the Gantt chart:-\n");
printf("Time-span-ms: %d\n", time-span-ms);
return 0;
}
```

Output:

Enter the number of tasks

Enter number of tickets for each task:

Task 1 tickets: 10

Task 2 tickets: 5

Task 3 tickets: 20

Running tasks:

Process Scheduling:

Time 0-1: Task 2 is running

Time 1-2: Task 3 is running,

time 2-3: Task 1 is running.

12/16/24

LAB-5 program.



Write a C program to stimulate producer-consumer problem using semaphores

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 5, x = 0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("in 1. producer in 2. consumer in 3. exit");
    while (1)
    {
        printf("in 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;
        }
    }
}
```

```
break;
case 3:
    exit(0);
    break;
}
3
return 0;
}

int wait(int s)
{
    return (-s);
}

int signal(int s)
{
    return (+s);
}

void producer()
{
    mutex = wait(mutex);
    full = signal(full);
    empty = wait(empty);
    x++;
    printf("in producer 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 - i just std::cout & cin & a file
mutex = signal (mutex) to handle the

output:

1. producer

2. consumer

3. Exit

Enter your choice : 1

producer produces the item 1

Enter the your choice : 1

producer produces the item 2

Enter the your choice : 1

producer produces the item 3

Enter your choice : 2

consumer consumes item 2

12/01/2022 10:50:40

19/6/24

1 AB-6 program.

→ write a C program to stimulate Banker's algorithm
for the purpose of deadlock avoidance

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

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

```
#define Max-processes 10
```

```
#define Max-Resources 10
```

```
bool isSafeState (int processes, int resources, int available)
```

```
int max[10][Max-Resources], int allocation[10]
```

```
[Max-Resources], int need[10][MAX-RESOURCES], int
```

```
safeSequence[]);
```

```
int main()
```

```
{
```

```
int processes, resources, ...;
```

```
int available[Max-Resources];
```

```
int max[Max-Processes][Max-Resources];
```

```
int allocation[Max-Processes][Max-Resources];
```

```
int need[Max-processes][Max-Resources];
```

```
int safeSequence[Max-Processes];
```

```
printf("Enter the number of processes:");
```

```
scanf("%d", &processes);
```

```
printf("Enter the number of resources:");
```

```
scanf("%d", &resources);
```

```
printf("Enter the available resources:\n");
```

```
for (int i=0; i<resources; i++)
```

```
{
```

```
scanf("%d", &available[i]);
```

```
printf("Enter the maximum resource matrix:\n");
```

```
for (int i=0; i<processes; i++)
```

```

$ cd C:\Users\H\Downloads\H20230909\OS\Notes
122

printf("Enter details for P1. \n", i);
for (int j=0; j<resources; j++)
{
    scanf("%d", &max[i][j]);
}

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

calculate need(processes, resources, max, allocation,
need);

if (isSafeState(processes, resources, available, max,
allocation, need, safeSequence))
{
    printf("SYSTEM IS IN SAFE STATE\n");
    printf("The safe sequence is - - (%d)\n",
    for (int i=0; i<processes; i++)
    {
        printf("%d", safeSequence[i]);
    }
    printf("\n");
}
else
{
    printf("System is not in safe state.\n");
}

printf("System is NOT IN SAFE STATE");
}

```

```

printf("In process i Allocation & It Max If It Need\n");
for(int i=0; i<processes; i++)
{
    printf("P-1. d It", i);
    for(int j=0; j<resources; j++)
    {
        printf(" + d allocation[i][j]");
    }
    printf("\n");
}
for(int i=0; i<processes; i++)
{
    printf("P-", i);
    for(int j=0; j<resources; j++)
    {
        printf(" d, max[i][j]");
    }
}
printf("\n");
return 0;
}

void calculateNeed(int processes, int resources, int max
    [MAX_RESOURCES], int allocation[MAX_RESOURCES][MAX_RESOURCES], int
    need[MAX_RESOURCES])
{
    for (int i=0; i<processes; i++)
    {
        for (int j=0; j<resources; j++)
        {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

```

```

bool isSafeState (int processes, int resources, int available),
int max[] [Max-Resources], int allocation[] [Max-Resources],
[Max-Resources], int need [] [Max-Resources], int
safe sequences[])
{
    int work [Max-Resources];
    bool finish [Max-Processes] = {false};
    int count = 0;
    for (int i=0; i < resources; i++)
    {
        work [i] = available [i];
    }
    while (count < processes)
    {
        bool found = false;
        for (int i=0; i < processes; i++)
        {
            if (!finish [i])
            {
                int canAllocate = true;
                for (int j=0; j < resources; j++)
                {
                    if (need [i][j] > work [j])
                    {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate)
                {
                    for (int j=0; j < resources; j++)
                    {
                        work [j] += allocation [i][j];
                    }
                    finish [i] = true;
                    count++;
                }
            }
        }
    }
}

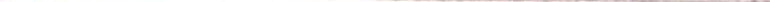
```

```
if (!finish[i]) return false; // If job i is not finished yet, return false.

SafeSequence found[i]; // Initialize found[i] to false.
found[i] = true; // Set found[i] to true.

printf("Process %d is visited.", i);
for (int j = 0; j < resources; j++) {
    printf("-%d", work[j]);
}
printf("\n");

if (found[i]) {
    for (int i = 0; i < processes; i++) {
        if (!finish[i]) {
            return false;
        }
    }
    return true;
}
return true;
```

Output: 

6.05 828 91.8 97

Enter the number of processes: 8

Enter the number of resources: 3

Enter the number of available resources: 2

332 .⁵⁰ 284 .⁵⁰ 0 VI

Enter the maximum response matrix

Enter details for P_0

753

Enter details for p, 12.000

Enter details for P2

902 *Adonibulus oblongus*

Enter details for P3

2 2 2 329347001117-0000000000000000

Enter details about [Puzzles at Work](https://www.puzzlesatwork.com) here:

6.1. H_2O and H_2S are two common acids.

~~Enter the allocation/resource matrix~~

~~610~~ ~~262~~ ~~610~~ ~~262~~

~~100~~ 19/6 11-513-46-6

211

163/163 total, built at

9.15 visited (E 3.3) 3

1976-1977 (3/23/77) 26 648 211420

Published (in my) communication at

So it's visited (98%) so it's not lost.

is visited (105-1)

System (Algorithmic, Statistical)

The code sequence is $B \rightarrow (P_1 P_3 P_4 P_0 P_2)$

~~Wetlands Management Plan~~

	Process Abstraction			Max	Next, after
P0	0	10	753	313	
P1	200	300	322	122	
P2	302	902	600	100	
P3	21100	2122	2111	1000	
P4	002	433	431	333	

① Main

Lt 8-7 Program

19/6/24 Dining Philosopher

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

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

```
#define MAX_PHILOSOPHERS
```

```
Void allow-one-to-eat(int hungry[], int n)
```

}

```
int isWaiting[MAX_PHILOSOPHERS];
```

```
for(int i=0; i<n; i++)
```

{

```
isWaiting[i]=1;
```

}

```
for(int i=0; i<n; i++)
```

{

```
printf("P-%d is granted to each eat in", hungry[i]);
```

```
isWaiting[hungry[i]] = 0;
```

```
for(int j=0; j<n; j++)
```

{

```
if(isWaiting[hungry[j]])
```

{

```
printf("P-%d is waiting in %d", hungry[j]);
```

7
2

column 3ii

for (int k=0; k<n; k++)

{ if (philosophers[k].x1==0 & & philosopher[k].x2==0) {
 if (philosophers[k].isWaiting[k]==1) {
 philosophers[k].isWaiting[k]=0; } } else {
 if (philosophers[k].isWaiting[k]==0) {
 philosophers[k].isWaiting[k]=1; } } }if (philosophers[i].x1==0 & & philosopher[i].x2==0) {
 if (philosophers[i].isWaiting[i]==0) {
 philosophers[i].isWaiting[i]=1; } } else {
 if (philosophers[i].isWaiting[i]==1) {
 philosophers[i].isWaiting[i]=0; } } }

void allowTwoToEat(int hungry[], int n)

{ if (n<2 || n>Max_Philosophers) {
 printf("Invalid number of philosophers.\n");
 return; }

for (int i=0; i<n-1; i++)

{ for (int j=i+1; j<n; j++)

{ if (hungry[i]>0 & & hungry[j]>0) {
 printf("P%d and P%d are granted to eat in", hungry[i], hungry[j]); } }

for (int k=0; k<n; k++)

{ if ((k!=i) & & (k!=j)) {
 printf("P%d is waiting in", hungry[k]); } }

printf("P%d is waiting in", hungry[k]);

{}

{}

{}

{}

int main()

{

int total_philosophers, hungry_count;

int hungry_positions[Max_Philosophers];

Printf ("Dining philosopher problem");

Printf ("Enter the total no. of philosophers");

scanf ("%d", &total_philosophers);

if (total_philosophers > Max_Philosophers) total_philosophers = Max_Philosophers;

return 1;

if (total_philosophers < 2) return 1;

Printf ("How many are hungry?");

scanf ("%d", &hungry_count);

if (hungry_count < 1 || hungry_count > total_philosophers)

return 1;

else printf ("Enter %d philosopher positions.", hungry_count);

return 1;

for (int i=0; i<hungry_count; i++)

{

printf ("Enter philosopher %d position: ", i+1);

scanf ("%d", &hungry_positions[i]);

if (hungry_positions[i] < 0 || hungry_positions[i] > total_philosophers)

{

printf ("invalid philosopher position.\n");

return 1;

}

int choice;

(whichever)

{

Print 1 in 1; one can eat at a time (n),
Print 1's - Two can eat at a time (1),
Print 1"3 - Exit n"}
print ("Enter your choice:");
scanf ("%d", &choice);
switch (choice){
case 1: // option to eat one at a time
allow_one_to_eat(hungry_positions, hungry_count);
break;
case 2: // option to eat two at a time
allow_two_to_eat(hungry_positions, hungry_count);
break;
case 3: // no opt chosen
default:
printf ("invalid choice");
}

}
} // if choice is 1 or 2
else if choice is 3
{
hungry_positions = 0;
hungry_count = 0;

Churn,

Enter total no of philosophers: 5
How many are hungry: 2
Enter philosopher 1 position: 1
Enter philosopher 2 position: 2

• one can eat at a time

2. Two can eat at a time

3. Exit

• One can eat at a time

2. Two can eat at a time

3. Exit

Enter your choice: ?

۳۱

LAB-Program

8) write a c program to simulate deadlock detection

#include <stdio.h>

void main()

int n,m,s;

int n, m, l);
printf("Enter the number of processes and number of
types of resources: int l);

sani ("y-d-t'd") (fn, fm)

int maxOn[m], needOn[m], allOn[m] // On[m], f[i]:=

finish [fɪnɪʃ] dead [deɪd] ice [aɪs], the art and culture

```
printf("Enter the maximum number of each type  
of resource needed by each process:\n");
```

-for (i=0; i<n; i++)

8

for (j=0; j<m ;j++)

۲

Scans "max[i][j]",

1

```
printf("Enter the allocated number of each typed  
resource needed by each process: m");
```

for($\text{FO}'', i \in 0; j + 1)$

9

for (j=0; j < m; i++)

1

For prints C's take the available number of each type
of resource: 3n²; for files, 3n.

~~for (j=0 ; j<m; j+1)~~

```
{  
    Scanf ("%d", &ava[j]);  
    }  
    for (i=0; i<n; i++)  
    {  
        for (j=0; j<m; j++)  
        {  
            need[i][j] = max[i][j] - ava[i][j];  
        }  
        for (i=0; i<n; i++)  
        {  
            finish[i] = 0;  
        }  
        while (fbg)  
        {  
            flag = 0;  
            for (i=0; i<n; i++)  
            {  
                c = 0;  
                for (j=0; j<m; j++)  
                {  
                    if (finish[i] == 0 && need[i][j] <= ava[j])  
                    {  
                        c++;  
                        if (c == m)  
                        {  
                            for (j=0; j<m; j++)  
                            {  
                                if (finish[i] == 0 && need[i][j] <= ava[j])  
                                {  
                                    finish[i]++;  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

{
avag[j][j] = avg[i][j];

finish[i][j] = 1;

flag = 1;

?
if (finish[i][j] == 1)

{

i = n;

}

}

}

}

}

j = 0;

flag = 0;

for (i = 0; i < n; i++)

{

if (finish[i][j] == 0)

{

dead[j][i] = 1;

j++;

flag = 1;

}

if (flag == 1)

{

printf("Dead lock has occurred:\n");

printf("The dead lock processes are: \n");

for (i = 0; i < n; i++)

{

```

        printf("Printed %d\n", i);
    }
}

else
{
    cout << "No deadlock occurred!\n";
}

```

Out put:

Enter the number of processes and number of types of resources:

5 3

Enter the maximum number of each resource needed by each process:

7 53

3, 2, 2

902

2 2 4

10-14-13 3 weeks from peak 157.4 miles

Gather the allocated number of each type of resources needed by each process.

— 6 —

200

1203

213

2

600

Digitized by srujanika@gmail.com

1888.]

~~11~~ 127 B.

1111000

- Q) write a c program to stimulate the following contiguous memory allocation techniques.

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

```
Struct Block {
```

```
{
```

```
int block-no;
```

```
int block-size;
```

```
int is-free;
```

```
};
```

```
Struct File {
```

```
int file-no;
```

```
int file-size;
```

```
};
```

```
void firstFit(Struct Block blocks[], int nBlocks, Struct File files[], int nFiles)
```

```
{
```

```
printf("Memory Management scheme - First Fit")
```

```
printf("File No : File Size : Block No : Block Size")
```

```
"Fragmentation");
```

```
for (int i = 0; i < nFiles; i++)
```

```
{
```

```
for (int j = 0; j < nBlocks; j++)
```

```
if (blocks[j].is-free == 1 && blocks[j].block-size == files[i].file-size)
```

```
{
```

```
blocks[j].is-free = 0;
```

```
printf("File No : %d File Size : %d Block No : %d Block Size : %d\n",
```

```
files[i].file-no, files[i].file-size, blocks[j].block-no,
```

```
blocks[j].block-size);
```

```
}
```

```

void worstFit(struct Block blocks[], int n_blocks, struct file
files[], int n_files) {
    cout << "Memory Management scheme - Worst Fit" << endl;
    cout << "File-no: " << file->size << " Block-no: " << block->size
        << " Fragment id:" << endl;
    for (int i=0; i<n_files; i++) {
        cout << endl;
        int worst_fit_block = -1;
        int max_fragment = -1;
        for (int j=0; j<n_blocks; j++) {
            if (!blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                if (fragment > max_fragment)
                    max_fragment = fragment;
                worst_fit_block = j;
            }
        }
        if (worst_fit_block == -1)
            cout << "No free block found" << endl;
        cout << "Worst-fit block: " << worst_fit_block << endl;
        cout << "Block size: " << blocks[worst_fit_block].block_no << endl;
        cout << "File size: " << files[i].file_size << endl;
        cout << endl;
    }
}

```

```

    void bestFit(struct block blocks[], int n_blocks,
    struct file files[], int n_files) {
        printf("Memory Management Scheme: Best Fit\n");
        printf("File No.: File Size: Block No.: Block Size\n");
        if (fragment != 0) {
            for (int i = 0; i < n_files; i++) {
                if (blocks[j].is_free & blocks[j].block_size == files[i].file_size) {
                    if (best_fit_block == -1) {
                        best_fit_block = j;
                    } else if (blocks[best_fit_block].block_size - files[i].file_size < min_fragment) {
                        min_fragment = fragment;
                        best_fit_block = j;
                    }
                }
            }
            if (best_fit_block != -1) {
                printf("%d %d %d %d\n", files[i].file_no, files[i].file_size, blocks[best_fit_block].block_no, blocks[best_fit_block].block_size);
                blocks[best_fit_block].is_free = 0;
                printf("File No.: File Size: Block No.: Block Size\n");
                printf("%d %d %d %d\n", files[i].file_no, files[i].file_size, blocks[best_fit_block].block_no, blocks[best_fit_block].block_size);
            }
        }
    }
}

```

```

int main()
{
    int n-blocks-on-files;
    printf("Enter the number of blocks:");
    scanf("%d",&n-blocks);
    printf("Enter the number of files:");
    scanf("%d",&n-files);
    struct Block blocks[n-blocks];
    for (int i=0; i<n-blocks; i++)
    {
        printf("Enter the size of block %d: ", i+1);
        scanf("%d", &blocks[i].block-size);
        blocks[i].is-free=1;
    }
    struct file files[n-files];
    for (int i=0; i<n-files; i++)
    {
        printf("Enter file no. %d: ", i+1);
        scanf("%d", &files[i].file-no);
        printf("Enter file size of file %d: ", i+1);
        scanf("%d", &files[i].file-size);
    }
    for (int i=0; i<n-blocks; i++)
    {
        if (files[0].file-no == 1)
        {
            blocks[0].is-free=0;
            blocks[0].block-no = 1;
            blocks[0].block-size = files[0].file-size;
        }
    }
}

```

`best_fit(blocks, n-blocks, n-frees)`

returno;

3

Chlorophyll a fluorescence

Distanz: 1077 km (ca. 10 h)

Sub the number of blocks: 3 = $\frac{1}{3}$ of 9 blocks

Enter the number of alleles: 2

enter the size of docx's

Enter the size of block: 2

~~Enter the size of blocks:~~

Enter the size of file 1: 1

Enter the `SECRET` file:

1980-1981 学年第一学期

Memory management schemes for main memory

Block size = E_n

4 3 7 2

120911 100000Z JUN 13

Memory management software - a brief note.

Filesize filesize in

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

~~2 4 3 7 6 11~~

1. *What is the meaning of the following?*

Memory management

FileNo : Filesize : Register Scheme - Bestfit.

Block No. 1, Blocksize 1F

$$2 \quad 4 \quad 2 \quad \dots + 2 + 1$$

~~666~~ 1 5 1

W. C. L. - 1900

www.english-test.net

 BOSTON PUBLIC LIBRARY

10/7/24

LAB-10 program

Date _____
Page _____

write a C-program to stimulate page replacement algorithms

a) FIFO

b) LRU

c) optimal

• efficient approach to handle memory allocation and deallocation

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

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

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

```
void fifo(int pages[], int n, int capacity)
```

```
{
```

```
    int frame[capacity], index = 0, page_faults = 0;
```

```
    for (int i = 0; i < capacity; i++)
```

```
        frame[i] = -1;
```

```
    for (int i = 0; i < n; i++)
```

```
{
```

```
        if (frame[index] == -1)
```

```
            frame[index] = pages[i];
```

```
        else
```

```
            for (int j = 0; j < capacity; j++)
```

```
{
```

```
                if (frame[j] == -1)
```

```
{
```

```
                    frame[j] = pages[i];
```

```
                    break;
```

```
                }
```

```
            }
```

```
            if (page_faults > 0)
```

```

printf("PF NO. %d", page-faults);
printf("\n");
}

void LIFO(int pages[], int n, int capacity)
{
    int frame[capacity], index=0, page-faults=0;
    for (int i=0; i<capacity; i++)
        frame[i]=-1;
    printf("FIFO page Replacement : process\n");
    for (int i=0; i<n; i++)
    {
        if (!found)
        {
            for (int j=0; j<capacity; j++)
                if (frame[j] == pages[i])
                    found=1;
        }
        if (!found)
        {
            frame[index]=pages[i];
            index=(index+1)%capacity;
            page-faults++;
        }
    }
    printf("Total page-faults using FIFO : %d\n", page-faults);
}

```



```
printf("Total page faults using FIFO: %d\n", page-faults);
}

void LRU(int pages[], int n, int capacity)
{
    int frame[capacity], counter[capacity], time=0;
    page-faults=0; // Initialize page fault count to 0
    for (int i=0; i<capacity; i++)
        frame[i]=-1; // Initialize frames to -1
    for (int i=0; i<n; i++)
        counter[i]=0; // Initialize counters to 0
    printf("LRU Page Replacement Process:\n");
    for (int i=0; i<n; i++)
    {
        int found=0; // Initialize found flag to 0
        for (int j=0; j<capacity; j++)
        {
            if (frame[j]==pages[i]) // If page is already in frame
                if (counter[j]>time) // If current time is greater than counter
                    found=1; // Set found flag to 1
            counter[j]=time+1; // Update counter for current page
        }
        if (!found) // If page is not found
        {
            int min=INT_MAX, min_index=-1;
            for (int j=0; j<capacity; j++)
            {
                if (counter[j]<min)
                    min=counter[j];
            }
            if (counter[frame[min_index]]>time)
                frame[min_index]=pages[i]; // Replace page in frame
            else
                page-faults++; // Increment page fault count
        }
    }
}
```

min = counter[ij];

min = index[ij];

}

frame[min-index] = pages[ij];

counter[min-index] = timeti;

Page-faults++;

}

print-frames(frame, capacity); found ? 0 : page-faults;

}

printf("Total page faults using LRU: %d\n", page-faults);

}

void optimal (int page[], int n, int capacity)

{

int frame[capacity], page-faults = 0;

for (int i=0; i<capacity; i++)

frame[i] = -1;

printf("Optimal page Replacement process:\n");

-for (int i=0; i<n; i++)

{

int found = 0;

-for (int j=0; j<capacity; j++)

{

if (frame[j] == page[i])

{

found = 1;

bread;

}

}



if (!found) {
 cout << "Page fault at address " << address << endl;

{

int farthest = i + 1; index = -1; i += 1;

for (int j=0; j<capacity; j++)

{

int k;

for (k=i+1; k<n; k++)

{

if (frame[j] == pages[k]) {

break;

} // If found, increment address and break loop

if (k > farthest) {

{

farthest = k; // Set farthest frame number

index = j; // Set index of farthest frame

{

if (index == -1) {

{

for (int j=0; j<capacity; j++) {

{

if (frame[j] == -1) {

{

if (pages[j] != blank) {

{

index = j;

{

break;

{

}

{

frame[index] = pages[i];

page_fault++;

{

```
print -> frames[frame, capacity, found? 0 : page_fault];
}

prints ("Total page faults using optimal ::", dimm);
P
}

int main()
{
    int n, capacity;
    printf ("Enter the number of pages: ");
    scanf ("%d", &n);
    int *pages = (int*) malloc (n * sizeof (int));
    printf ("Enter the pages: ");
    for (int i = 0; i < n; i++)
        scanf ("%d", &pages[i]);
    printf ("Enter the frame capacity: ");
    scanf ("%d", &capacity);
    printf ("in pages: ");
    for (int i = 0; i < n; i++)
        printf ("%d ", pages[i]);
    printf ("in %n");
    if (sfo (pages, n, capacity));
    lru (pages, n, capacity);
    optimal (pages, n, capacity);
    free (pages);
    return 0;
}
```



Output:

Enter the number of pages: 20

Enter the pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 0 1

8 0 1 1 1 4 1 0

Enter the frame capacity: 3

pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

FIFO Page Replacement process:

7 - PF No. 1

7 0 - PF No. 2

1 0 1 1 1 1 1 1 1 1

7 0 1 PF No. 3

1 0 1 1 1 1 1 1 1 1

2 0 1 PF No. 4

1 0 1 1 1 1 1 1 1 1

2 0 1

1 0 1 1 1 1 1 1 1 1

2 3 1 PF No. 5

1 0 1 1 1 1 1 1 1 1

2 3 0 PF No. 6

1 0 1 1 1 1 1 1 1 1

4 3 0 PF No. 7

1 0 1 1 1 1 1 1 1 1

4 2 0 PF No. 8

1 0 1 1 1 1 1 1 1 1

4 2 3 PF No. 9

1 0 1 1 1 1 1 1 1 1

0 2 3 PF No. 10

1 0 1 1 1 1 1 1 1 1

0 2 3

1 0 1 1 1 1 1 1 1 1

0 1 3 PF No. 11

1 0 1 1 1 1 1 1 1 1

0 1 2 PF No. 12

0 1 2

7 1 2 PF No. 13

7 0 2 PF No. 14

7 0 1 PF No. 15

Total page faults using FIFO: 15

LRU page Replacement process:

1 - PF No. 1

0 → PF No. 2

0 → PF NO. 3

0 1 2 PF NO. 4

0 1 2 - PF NO. 5

0 3 2 PF NO. 6

0 3 2

0 3 4 PF NO. 7

0 2 4 PF NO. 8

3 2 4 PF NO. 9

3 2 0

3 2 1 PF NO. 10

3 2 1

0 2 1 PF NO. 11

0 2 1

0 7 1 PF NO. 12

0 7 1

0 7 1

Total page faults using LRU: 12



Optimal Page Replacement process:

3 - PF No. 1

20 - PF No. 2

20 1 PF No. 3

20 1 PF No. 4

20 1

20 3 PF No. 5

20 3

20 3 PF No. 6

20 3

20 3

20 3 PF No. 7

20 3

20 3

20 1 PF No. 8

20 1

20 1

20 1

40 1 PF No. 9

40 1

40 1

Total page fault using optimal: 9

40 1
30 1
20 1