

INDEX

Name nallabothula sai srujan (1BM22CS170)

Sub.

Std.:

Div.

Roll No.

Telephone No.

E-mail ID.

Blood Group.

Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
1	8/5/2024 a) FCFS		
	b) SJF non preemptive		
	c) preemptive		
2	15/5/2024 a) priority		
	b) round robin		
3	22/5/2024 multilevel queue		
4	5/6/2024 a) rate monotonic scheduling		
	b) earliest deadline first scheduling		
	c) proportionality scheduling		
5	12/6/2024 producer-consumer problem		
	using semaphores		
6	19/6/2024 a) bankers algorithm for the		
	purpose of deadlock avoidance		
7	19/6/2024 b) dining philosopher		
8	3/7/2024 deadlock detection		
9	3/7/2024 memory allocation		
	a) worst fit		
	b) best fit		
	c) first fit		
10)	10/7/2024 page replacement algorithms		
	1) FIFO		
	2) LRU		
	3) Optimal		

Scheduling Programming

Scheduling Programming

Lab-1 8/5/24

- (1) write a c program to simulate the following non preemptive cpu scheduling algorithm
algorithm 2 find turn around time and waiting time

(i) FCFS

```
#include <stdio.h>
int main(){
    int n, i;
    float atime = 0, awt = 0;
    printf("enter the number of process");
    scanf("%d", &n);
    int atime[n], btime[n], chime3[n], tatime4[n], wtimes[n];
    printf("enter arrival time of process");
    for(i=0; i<n; i++){
        scanf("%d", &atime[i]);
    }
    printf("enter burst time of process");
    for(i=0; i<n; i++){
        scanf("%d", &btime[i]);
    }
    for(i=0; i<n; i++){
        if(i==0){
            chime3[i] = atime[i] + btime[i];
        }
        else{
            if(chime3[i-1] < atime[i]){
                chime3[i] = (atime[i] - chime3[i-1]) + chime3[i-1] + btime[i];
            }
            else{
                chime3[i] = chime3[i-1] + btime[i];
            }
        }
    }
    for(i=0; i<n; i++){
        tatime4[i] = chime3[i] - atime[i];
    }
}
```

```

}
for(i=0; i<n; i++){
    wtimes[i] = tattime4[i] - btime2[i];
}

for(i=0; i<n; i++){
    atat = atat + tattime4[i];
}

atat = (atat/n);

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

awt = (awt/n);

for(i=0; i<n; i++){
    printf("process id %d arrival time %d burst time %d complete
          time %d turn around time %d waiting time %d \n", i,
          atime1[i], btime2[i], ctime3[i], tattime4[i], wtimes[i]);
}

printf("average turn around time is %.f", atat);
printf("average working time is %.f", awt);
}

```

Output:

enter the number of process4
enter arrival time of process 0

1

5

6

enter burst time of process 2

2

3

4

process id 1 arrival time 0 burst time 2 complete time 2 turn around time 2
waiting time 0

process id 2 arrival time 1 burst time 2 complete time 4 turn around time 3
Waiting time 1

Process id 3 arrival time 5 burst time 3 complete time 8 turn around time 3
Waiting time 0

process id 4 Arrival time 6 burst time 4 complete time 12 turn around time 6
waiting time 2

average

average turn around time

SJF (shortest job first)

non preemptive

(b) SJF (shortest job first)

non preemptive

1 #include <stdio.h>

int main()

int n, i, j, temp1, temp2;

float atat = 0, awt = 0;

printf(" enter the numbe

scanf("%d", &n);

int atime[n], btime2

printf(" enter process

printf(" enter process

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

scanf("%d", &pid

);

printf(" enter arrival t

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

scanf("%d", &atime

);

printf(" enter burst t

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

scanf("%d", &btime

);

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

for(j=0; j<n-i-1;

if (btime2[j]>

temp1 = bt

btime2[j]

btime2[j+1]

temp2 = b

temp1 = b

</

average turn around time is 3.50000 average waiting time is 0.75000

```
#include<stdio.h>
int main(){
    int n,i,j,temp1,temp2;
    float tot=0, awt=0;
    printf(" enter the number of process ");
    scanf("%d", &n);
    int atime1[n], btime2[n], chime3[n], tattime4[n], wtimes[n], pid[n];
    printf(" enter process id's ");
    printf(" enter arrival times ");
    for(i=0; i<n; i++){
        scanf("%d", &pid[i]);
    }
    printf(" enter burst times ");
    for(i=0; i<n; i++){
        scanf("%d", &btime2[i]);
    }
    for(i=0; i<n-1; i++){
        for(j=i+1; j<n; j++){
            if(btime2[j] > btime2[i]){
                temp1 = btime2[i];
                btime2[i] = btime2[j];
                btime2[j] = temp1;
                temp2 = pid[i];
                pid[i] = pid[j];
                pid[j] = temp2;
            }
        }
    }
    for(i=0; i<n; i++){
        awt += (atime1[i] - pid[i]) / btime2[i];
        tot += btime2[i];
    }
    awt = awt / n;
    tot = tot / n;
    printf(" turn around time ");
    for(i=0; i<n; i++){
        printf("%d ", tattime4[i]);
    }
    printf("\n");
    printf(" waiting time ");
    for(i=0; i<n; i++){
        printf("%d ", wtimes[i]);
    }
    printf("\n");
}
```

```

}
for(i=0; i<n; i++){
    if(i==0){
        ctime3[i] = atime1[i] + btime2[i];
    }
    else{
        if(ctimes[i-1] < atime1[i]){
            ctimes[i] = (atime1[i] - ctimes[i-1]) + ctime3[i-1] +
                btime2[i];
        }
        else{
            ctime3[i] = ctime3[i-1] + btime2[i];
        }
    }
}

for(i=0; i<n; i++){
    tattime4[i] = ctimes[i] - atime1[i];
}

for(i=0; i<n; i++){
    wtimes[i] = tattime4[i] - btime2[i];
}

for(i=0; i<n; i++){
    atat = atat + tattime4[i];
}

atat = (atat/n);

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

awt = (awt/n);

for(i=0; i<n; i++){
    printf("process id %d arrival time %d burst time %d complete
time %d turnaround time %d waiting time %d\n",
pid[i], atime1[i], btime2[i], ctime3[i], tattime4[i],
wtimes[i]);
}

```

```

}
printf("average turn around time : %f", awt);
printf("average waiting time : %f", awt);
}

int output();
enter the number of processes : 4
enter process ids : 1 2 3 4
enter arrival times : 0 0 0 0
enter burst times : 6 3 3 4
process id 4 arrival time : 0
process id 1 arrival time : 0
process id 3 arrival time : 0
process id 2 arrival time : 0
average turnaround time : 3.5
average waiting time : 1.5
#include<stdio.h>
#define max 10
void sif(int n, int at[])
{
    int ct[max];
    int tat[max];
    int wr[max];
    int total_wt;
    int total_wt=0;
    int total_tat=0;
    for(int i=0; i<n; i++)
    {
        ct[i]=-1;
    }
}

```

```

        wtimes[i]);
    }
    printf("average turnaround time %f\n", atat);
    printf("average waiting time %f\n", awt);
    return 0;
}

int main()
{
    int n, i;
    float at[10], bt[10], tat[10], wt[10];
    float atat = 0, awt = 0;
    cout << "enter the number of processes ";
    cin >> n;
    cout << "enter process ids ";
    for(i = 0; i < n; i++)
    {
        cout << "process id " << i+1 << endl;
        cout << "enter arrival times ";
        cin >> at[i];
        cout << "enter burst times ";
        cin >> bt[i];
    }
    for(i = 0; i < n; i++)
    {
        atat += at[i];
        awt += (tat[i] - at[i]) / n;
    }
    cout << "average turnaroundtime " << atat;
    cout << "average waiting time " << awt;
}

#include<stdio.h>
#define max 10
void SJF(int n, int at[], int bt[])
{
    int ct[max];
    int tat[max];
    int wt[max];
    int total_wt;
    int total_tat = 0;
    for(int i=0; i<n; i++)
    {
        ct[i] = -1;
    }
}

```

Time %d complete
ting time %d\n",
[i], tattime4[i],

```
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        if(bt[j] > bt[j+i])
            int temp = bt[j];
            bt[j] = bt[j+i];
            bt[j+i] = temp;
            int temp_at = at[i];
            at[i] = at[j+i];
            at[j+i] = temp_at;
```

```
for(int i=0; i<n; i++)
    if(current_time < at[i])
        current_time = at[i];
    ct[i] =
```

1000

~~SJF (shortest job first)~~

~~Preemptive~~

```
#include <stdio.h>
#define max 10
int find_min(int arr[], int n){
    int min = arr[0];
    int index = 0;
```

```
for(int i=0; i<n; i++)
    if(arr[i] < min){
        min = arr[i];
        index = i;
    }
return index;
```

void SJF_Preemptive(int n, int

```
int ct[max] = {0};
int tat[max] = {0};
int wt[max] = {0};
int rt[max];
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;
int completed_processes;
while(completed_processes <
```

```
int available_processes;
int available_processes;
for(int i=0; i<n; i++)
    if(at[i] <= current_time)
        available_processes++;
        available_processes++;
        available_processes++;
        available_processes++;

if(available_processes == n)
    current_time = at[n-1];
```

```

for(int i=0; i<n; i++){
    if(arr[i] < min){
        min = arr[i];
        index = i;
    }
}
return index;
}

void SJF_Preemptive(int n, int at[], int bt[]){
    int ct[max] = {0};
    int tat[max] = {0};
    int wt[max] = {0};
    int rt[max];
    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;
    int completed_processes = 0;
    while(completed_processes < n){
        int available_processes[max];
        int available_count = 0;
        for(int i=0; i<n; i++){
            if(at[i] <= current_time && rt[i] > 0){
                available_processes[available_count] = i;
                rt[i] = rt[i] - 1;
                available_count++;
            }
        }
        if(available_count == 0){
            current_time++;
        }
    }
}

```

```

        continue;
    }

    int shortest_job_index = available_processes[find_min(rt, + available_time)];
    rt[shortest_job_index] -= current_time++;
    if(rt[shortest_job_index] == 0) {
        completed_processes++;
        ct[shortest_job_index] = current_time;
        tat[shortest_job_index] = ct[shortest_job_index] - at[shortest_job_index];
        wt[shortest_job_index] = tat[shortest_job_index] - bt[shortest_job_index];
        total_wt += wt[shortest_job_index];
        total_tat += tat[shortest_job_index];
    }
}

printf("\n process \t arrival time \t burst time \t completion time \t turnaround \n
       . time \t waiting time \n");
for(int i=0; i<n; i++) {
    printf("%d\t %d \t %d \t %d \t %d \t %d \n", it[i],
          at[i], bt[i], ct[i], tot[i], wt[i]);
}

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

int main(){
    int n;
    printf("enter the number of processes ");
    scanf("%d", &n);
    int at[max], bt[max];
    int ot[max], bt[max];
}

```

```

printf("enter the burst
for(int i=0; i<n; i++)
printf("enter the arr
for(int i=0; i<n; i++)
scanf("%d", &arr[i]);
}
printf("enter the burst
for(int i=0; i<n; i++)
scanf("%d", &bt[i]);
}
sjf-preemptive(n, arr, bt);

```

~~Output~~
Output
enter the number of process
Output +
enter the number of processes
enter the arrival time

2
 1
 4
 0
 2
 enter the burst time
 1
 5
 1
 6
 3

```

printf("enter the burst time\n");
for(int i=0;
    printf("enter the arrival time\n");
    for(int i=0; i<n; i++){
        scanf("%d", &at[i]);
    }
    printf("enter the burst time\n");
    for(int i=0; i<n; i++){
        scanf("%d", &bt[i]);
    }
    SJF-preemptive(n, at, bt);
}

```

~~output~~

~~enter the number of processes~~

~~output~~

~~enter the number of process 5~~

~~enter the arrival time~~

2
1
4
0
2
enter the burst time

process	arrival time	burst time	completion time	turnaround time	waiting time
1	2	1	3	1	0
2	1	5	7	6	1
3	4	1	8	7	3
4	0	6	13	13	7
5	2	3	16	14	11

~~average waiting time 4/10~~

```

    ct[0] = at[0] + bt[0];
    tot[0] = ct[0] - at[0];
    wt[0] = tot[0] - bt - copy[0];
    total_wt += wt[0];
    total_tat += tot[0];
    for(i=1; i<n; i++)
    {
        ct[i] = ct[i-1] + bt[i];
        tot[i] = ct[i] - at[i];
        wt[i] = tot[i] - bt - copy[i];
        total_wt += wt[i];
        total_tat += tot[i];
    }
    printf("\n process \t arrival time \t burst time \t priority \t completion time
           \t turnaroundtime \t waiting time\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               i+1, at[i], bt - copy[i], p[i], ct[i], tot[i], wt[i]);
    }
    printf("\n average waiting time %.2f", (float)total_wt/n);
    printf("\n average turnaroundtime %.2f", (float)total_tat/n);
}

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

```

```

int at[max], bt[max], p[max];
printf("enter the arrival time\n");
for(int i=0; i<n; i++){
    scanf("%d", &at[i]);
}
printf("enter the burst time\n");
for(int i=0; i<n; i++){
    scanf("%d", &bt[i]);
}
printf("enter the priority\n");
printf("enter the priority\n");
for(int i=0; i<n; i++){
    scanf("%d", &p[i]);
}
priority_non_preemptive(n, at, bt, p);

```

Output:

enter the number of process 4
 enter the arrival time

0
 1
 2
 3
 4
 enter the burst time
 5
 4
 2
 1
 enter the burst time
 5
 4
 2
 1

1
 enter the priority
 10
 20
 30
 40
 process arrival time burst time
 1 4 5
 2 2 4
 3 1 2
 4 3 1
 average waiting time 5.00
 average turnaround time 8.00

Enter the priority

10
20
30
40
50

process	arrival time	bursttime	priority	tat	wt	ct
1	4	5	40	1	4	5
2	2	4	30	5	1	7
3	1	2	20	10	8	11
4	0	1	10	16	15	16

average waiting time 5.00

average turnaround time 8.00

(b) Experiment with different quantum sizes for n algorithm

2) Round robin scheduling:

```
#include <stdio.h>
struct process {
    int pid;
    int burst_time;
    int arrival_time;
    int remaining_time;
};

void roundrobin(struct process processes[], int n, int time_quantum) {
    int remaining_processes = n;
    int current_time = 0;
    int completed[n];
    int ct[n], wt[n], tat[n], rt[n];
    for (int i = 0; i < n; i++) {
        completed[i] = 0;
    }
    while (remaining_processes > 0) {
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time == current_time) {
                if (completed[i] == 0 && processes[i].arrival_time <= current_time) {
                    if (processes[i].remaining_time > 0) {
                        if (processes[i].remaining_time <= time_quantum) {
                            current_time += processes[i].remaining_time;
                            processes[i].remaining_time = 0;
                            completed[i] = 1;
                            remaining_processes--;
                            ct[i] = current_time;
                            tat[i] = ct[i] - processes[i].arrival_time;
                        } else {
                            current_time += time_quantum;
                            processes[i].remaining_time -= time_quantum;
                        }
                    }
                }
            }
        }
    }
}
```

algorithm)

```
    }
    wt[i] = ct[i] - processes[i].arrival_time - processes[i].burst_time;
    rt[i] = wt[i];
}

for (int i=0; i<n; i++) {
    printf("pid|at|bt|ct|wt|tat|rt\n");
    float avg_tat=0, avg_wt=0;
    for (int j=0; j<n; j++) {
        printf("%d|%d|%d|%d|%d|%d|%d\n", processes[i].pid, processes[i].arrival_time, processes[i].burst_time, ct[j], wt[j], tat[j], rt[j]);
        avg_tat += tat[i];
        avg_wt += wt[i];
    }
    avg_tat /= n;
    avg_wt /= n;
    printf("average turnaround time %.2f\n", avg_tat);
    printf("average waiting time %.2f\n", avg_wt);
}

int main() {
    int n, time_quantum;
    printf("enter the number of processes");
    scanf("%d", &n);
    printf("enter the Time quantum");
    scanf("%d", &time_quantum);
    struct process processes[n];
    quantum;
```

```

printf("enter arrival time and burst time for each process\n");
printf(" enter arrival time and burst time for each process\n");
for(int i=0;i<n;i++){
    printf("enter arrival time for process %d ",i+1);
    scanf("%d",&processes[i].arrival-time);
    printf("enter burst time for process %d ",i+1);
    scanf(" %d",&processes[i].burst-time);
    processes[i].pid = i+1;
    processes[i].remaining-time = processes[i].burst-time;
}
roundrobin(processes,n,time quantum);
}

```

OUTPUT:

enter the number of processes 5
 enter the time quantum 2
~~enter arrival time and burst time for each pr~~
 enter arrival time and burst time for each process
 enter arrival time for Process 1 0
 enter burst time for process 1 5
 enter arrival time for process 2 1
 enter burst time for process 2 3
 enter arrival time for process 3 2
 enter burst time for process 3 1
 enter arrival time for process 4 3
 enter burst time for process 4 2
~~enter arrival time for process 5 4~~
~~enter burst time~~
 enter burst time for process 5 3

pid	at	bt	ct
1	0	5	14
2	1	3	12
3	2	1	5
4	3	2	7
5	4	3	13

~~average turnaroundtime 8.2~~
~~average turnaroundtime 8.2~~
~~average turnaroundtime 8.2~~
~~average waitingtime 5.4~~

average turnaround time 8
 average waiting time 5.4

~~earliest deadline first sche~~
~~earliest deadline first scheduling~~

```

#include<stdio.h>
typedef struct {
    int capacity;
    int deadline;
    int period;
}task;
int compare_tasks(const void*
)
task* task1 = (task*)
task* task2 = (task*)
return (task1->dead
}
int main(){
    int num_tasks;
}

```

pid	at	bt	ct	wt	tat	rt
1	0	5	14	9	14	9
2	1	3	12	8	11	8
3	2	1	5	2	3	2
4	3	2	7	2	4	2
5	4	3	13	6	9	6

average turnaround time = 8.20

average turnaround time = 8.20

average waiting time

average waiting time = 5.40

average turnaround time = 8.20

average waiting time = 5.40

earliest deadline first sche

earliest deadline first scheduling

```
#include<stdio.h>
typedef struct{
    int capacity;
    int deadline;
    int period;
}
```

task;

```
int compare_tasks(const void *a, const void *b){
```

task*

task1 = (task*)a;

task* task2 = (task*)b;

return (task1->deadline - task2->deadline);

```
}int main(){
```

int num_tasks;

3) ~~lab-3~~ write a C program to simulate the multilevel queue scheduling algorithm.

#include <stdio.h>

void findwaitingtime(int process[], int n, int bt[], int qt[], int wt[])

{

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

 {
 wt[i] = bt[i-1] + wt[i-1] - qt[i-1];

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

}

}

}

}

Void findturnaroundtime(int process[], int n, int bt[], int wt[], int tat[])

int tat[]

{

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

 tat[i] = bt[i] + wt[i];

}

}

}

~~your program~~

void roundrobin(int proce

{

 int wt[n], tat[n], ct[n],

 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)

 {
 if (time + quantum > remaining_bt[i])

 time = remaining_bt[i];

 if (remaining_bt[i] > time + quantum)

 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;
 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

 completed++;

 }

 else
 {
 time += quantum;

 remaining_bt[i] = time;

 ct[i] = time;

```

void roundrobin(int process[], 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 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 t=0; t<n; t++)
        {
            if (remaining_bt[t] > 0 & at[t] < time)
            {
                if (remaining_bt[t] > quantum)
                {
                    time += remaining_bt[t];
                    remaining_bt[t] = 0;
                    ct[t] = time;
                    completed++;
                }
                else
                {
                    time += quantum;
                    remaining_bt[t] = quantum;
                    ct[t] = time;
                    completed++;
                }
            }
        }
    }
}

```

```

findWaitingTime(process, n, bt, at, wt);
findTurnaroundTime(process, n, bt, wt, tat);

for(int i=0; i<n; i++)
{
    printf(" P %d |t %d|t %d|t %d|t %d\n", process[i],
           bt[i], at[i], wt[i], tat[i]);
}

total_wt += wt[i];
total_tat += tat[i];
total_bt += bt[i];
total_tat += tat[i];
total_wt += wt[i];
total_tat += tat[i];

printf(" average waiting time (round robin) = %.2f\n", (float) total_wt/n);
printf(" average turnaround time (round robin) = %.2f\n", (float) total_tat/n);

```

```

void fcfs(int process[], int n
{
    int wt[n], tat[n], ct[n];
    findWaitingTime();
    findTurnaroundTime();
    printf(" process burst time arrival time waiting time turnaround time
completion 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\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 \n");
    printf(" average turnaround time \n");
}

int main()
{
    int process[] = {1, 2, 3, 4, 5};
    int n = sizeof(process) / sizeof(int);
    int bt = {10, 5, 8, 12, 15};
    int at[] = {0, 1, 2, 3, 4};
    int quantum = 3;
}

```

```

void FCFS()

void FCFS(int process[], int n, int bt[], int at[])
{
    int wt[n], tat[n], ct[n];
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    findWaitingTime();
    findTurnaroundTime();
    printf(" process bursttime arrivaltime waitingtime turnaroundtime
           completion time \n");
    for (int i=0; i<n; i++)
    {
        ct[i] = at[i] + bt[i];
        printf(" %d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               process[i],
               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 process[] = {1, 2, 3, 4, 5};
    int n = sizeof(process)/sizeof(process[0]);
    int bt[] = {10, 5, 8, 12, 15};
    int at[] = {0, 1, 2, 3, 4};
    int quantum = 2;
}

```

~~int quantum~~

~~int quantum=2;~~

~~int quantum=2;~~

~~roundrobin (process, n, bt, at, quantum);~~

~~fdfs (process, n, bt, at);~~

~~}~~

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

average wt

Average wt (Round robin) = 14.60

average turnaround time = 20

average turnaround time (Round robin) = 24.60

process	bt	at	wt	tat	ct
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 wt (FCFS) = 14.60

average tat (FCFS) = 24.6

(a) y
rate monotonic

#include<stdio.h>

struct process {

int execution_time;

int time_periods;

};

int

int lcm(int a, int b)

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

while(1)

{ if(max%a==0&&max%b==0)

return max;

} else max+=lcm(a,b);

return(max);

}; return(max);

max+=t;

}

}

int is_schedulable(struct P

{ float utilization=0.0;

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

{ utilization += (float)

process[i].Time / period;

} return (utilization<=1.0);

};

int main()

{ struct process processes[5];

int main()

{ struct processes processes

{ 3,20};

{2,5};

{1,10};

```

int lcm(int a, int b)
{
    int max = (a>b)?a:b;
    while(1)
    {
        if(max%a==0 && max%b==0)
            return max;
        if(max%a==0 || max%b==0)
            return (max+1);
        max++;
    }
}

int is_schedulable(struct process processes[], int n)
{
    float utilization = 0.0;
    for(int i=0; i<n; i++)
    {
        utilization += (float)process[i].execution_time;
        if(process[i].time> period)
            return (Utilization >= 1.0);
    }
}

int main()
{
    struct process processes[2];
    int main()
    {
        struct processes processes[2] =
        {{3,20}, {2,5}};
    }
}

```

```

process
{2,10};

};

int n = sizeof(process) / sizeof(process[0]);
if (!is_schedulable(processes, n))
{
    printf("the given set of processes is not schedulable\n");
    return 0;
}

int scheduling_time = lcm(processes[0].time_period, processes[1].time_period);
printf("execution order\n");
if (processes[0].time_period == 0)
    printf("P0\n");
if (processes[1].time_period == 0)
    printf("P1\n");
if (processes[2].time_period == 0)
    printf("P2\n");
if (processes[3].time_period == 0)
    printf("P3\n");
}

```

```

}

```

~~output~~
execution order

~~output~~
execution order
~~P1 P2 P3~~

~~output~~
execution order

P2 P3 P1 P2 P2 P3 P2

(b) earliest deadline first scheduling

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort(int proc[], int d[])
{
    int temp = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (d[j] < d[i])
            {
                temp = d[i];
                d[i] = d[j];
                d[j] = temp;
            }
        }
    }
}

```

```

d[i] = temp;
temp = pt[i];
pt[i] = pt[j];
pt[j] = temp;
temp = b[i];
temp = b[j];
b[j] = b[i];
b[i] = temp;
temp = proc[i];
proc[i] = proc[j];
proc[j] = temp;
}
}
}
int gcd();
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++)
    {
        int lcm = p[0];
        for(int j=1; j<n; j++)
        {
            if(p[j] % lcm == 0)
                lcm *= p[j];
        }
        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];
    printf(" enter the cpu burst times");
    for(int i=0; i<n; i++)
    {
        scanf("%d", &b[i]);
        rem[i] = b[i];
    }
    printf(" enter the deadline");
    for(int i=0; i<n; i++)
    {
        scanf("%d", &d[i]);
    }
    printf(" enter the home position");
    for(int i=0; i<n; i++)
    {
        proc[i] = i+1;
    }
    sort(proc, d, b, pt, n);
    int l = lcmul(pt, n);
    printf(" earliest deadline first scheduling");
    for(int i=0; i<n; i++)
    {
        printf("%d\t", i+1);
        printf("%d\t", pt[i]);
    }
}

```

```

lcm = (lcm * p[i]) / gcd(lcm, p[i]);
}

return lcm;
}

void main()
{
    int n;
    printf(" enter the number of process ");
    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++)
    {
        proc[i] = i+1;
    }
    sort(proc, d, b, pt, n);
    int l = lcmul(pt, n);
    printf(" earliest deadline scheduling \n ");
    printf(" pid\tburst\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", n);
int time = 0;
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 < 1)
{
    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 = dt[1];
    int task_to;
    int tasktoexecute = -1;
    for(int i=0; i<n; i++)
    {
        if(rem[i] > 0 && nextdeadlines[i] < mindeadline)
        {
            mindeadline = nextdeadlines[i];
            tasktoexecute = i;
            tasktoexecute = i;
        }
    }
    if(tasktoexecute != -1)
    {
        printf("%dms : task %d is running\n", time, proc[tasktoexecute]);
    }
}

```

rem[tasktoexecute]

```

}
else
{
    printf("%dms : CPU is
    printf("%dms : CPU is
}
}
time++;
}
}

```

Output
enter the number of process 3
enter cpu burst time

3
2
2
enter the deadlines:
7
4
8
~~enter~~

enter the timeperiod
20
5
10

earliest deadline scheduling

Pid	burst	deadlines	pe
2	2	4	
1	3	4	
3	2	8	

Scheduling occurs for 20ms

0ms: task 2 is running

1ms: task 2 is running

2ms: task 1 is running

3ms: task 1 is running

4ms: task 1 is running

5ms: task 3 is running

6ms: task 3 is running

7ms: task 2 is running

8ms: task 2 is running

9ms: CPU is idle

```

rem[tasktoexecute]--;

if(eve)
{
    printf("%dms - CPU is idle\n");
    printf("%dms - CPU is idle \n", time);
}

// Time++
time++;
}

Output
enter the number of process 3
enter CPU burst time
3
2
enter the deadlines
7
4
8
enter the time period
20
5
10
earliest deadline scheduling
pid   burst  deadlines  period
2     2       4           5
1     3       7           20
3     2       8           10

Scheduling occurs for 20ms
0ms: task 2 is running
1ms: task 2 is running
2ms: task 1 is running
3ms: task 1 is running
4ms: task 1 is running
5ms: task 3 is running
6ms: task 3 is running
7ms: task 2 is running
8ms: task 2 is running
9ms: CPU is idle

```

```

10 ms task 2 is running
11 ms task 2 is running
12 ms task 3 is running
13 ms task 3 is running
14 ms CPU is idle
15 ms task 2 is running
16 ms task 2 is running
17 ms CPU is idle
18 ms CPU is idle
19 ms CPU is idle

```

(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;
    int total_ticks = 0;
    for (int i = 0; i < num_tasks; i++) {
        total_tickets += tasks[i].tickets;
    }
    srand(time(NULL));
    int current_time = 0;
    int completed = 0;
    int c;
    int completed_tasks = 0;
    printf("process scheduling\n");
    while (completed_tasks < num_tasks) {
        int winning_ticket = rand() % total_tickets;
    }
}

```

```

int cumulative_ticks = 0;
for (int i = 0; i < num_tasks; i++) {
    cumulative_ticks += tasks[i].tickets;
}
if (winning_ticket <= cumulative_ticks) {
    if (winning_ticket == cumulative_ticks) {
        if (current_time == 0) {
            current_time = 1;
        } else {
            current_time += time_unit_duration_ms;
        }
        break;
    }
    completed_tasks++;
}
current_time += time_unit_duration_ms;
*time_span_ms = current_time;
}

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\n");
        return 1;
    }
}

```

```

int cumulative_ticks = 0;
for(int i=0; i<num_tasks; i++) {
    cumulative_ticks += tasks[i].ticks;
    if(winning_ticket < cumulative_ticks) {
        winning_ticket = cumulative_ticks;
    }
}
if(winning_ticket < cumulative_ticks) {
    printf("time %d - %d task %d is running\n",
          current_time, (current_time - task[i].tid));
    current_time = current_time + task[i].duration;
    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 number between
              1 and %d\n", max_tasks);
        return(1);
    }
}

```

```

printf("enter\n");
printf("enter number of tickets for each task \n");
for(int i=0; i<num-tasks; i++)
    for(int i=0; i<num-tasks; i++)
        tasks[i].tid = i+1;
    printf("task %d ticket", tasks[i].tid);
    scanf("%d", &tasks[i].tickets);
}
printf("\n running tasks \n");
schedule(tasks, num-tasks, 8*time-span-ms);
printf("\n time span of the gantt chart: %d milliseconds\n");
printf("\n time span of the gantt chart %d milliseconds\n");
}

```

OUTPUT

enter the number of tasks 3

enter the number of tickets for each task

task 1 tickets 10

task 2 tickets 20

task 3 tickets 30

running tasks

~~process scheduling~~

~~process scheduling~~

~~process scheduling~~

process scheduling

time 0-1 task 3 is running

~~task~~

time 2-2 task 3 is running

time 2-3 task 1 is running

~~task~~

time span of gantt chart

time span of gantt chart

12-06-2024

write a c program to
Semaphores

```

#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n producer\n");
    while(1)
    {
        printf("In enter your choice\n");
        printf("In enter your choice\n");
        scanf("%d", &n);
        switch(n)
        {
            case 1 : if(mutex == 1)
            case 1 : if((mutex == 1) && (full == 0))
            {
                mutex = 0;
                full = 1;
                printf("producer inserted item\n");
            }
            else {
                printf("producer is waiting\n");
                wait(mutex);
            }
        }
        break;
        case 2 : if((mutex == 0) && (full == 1))
        {
            mutex = 1;
            full = 0;
            printf("consumer removed item\n");
        }
        else {
            printf("consumer is waiting\n");
            wait(mutex);
        }
    }
}

```

time span of gantt chart.

time span of gantt chart 300 milliseconds

12-06-2024

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("\n 1.producer\n 2.consumer\n 3.exit");
    while(1)
    {
        printf("\n enter your choice");
        printf("\n enter your choice");
        scanf("%d", &n);
        switch(n)
        {
            case 1 : if((mutex == 1) && (empty == 0))
            case 1 : if((mutex == 1) && (empty == 0))
            {
                producer();
            }
            else
            {
                printf("buffer full");
            }
            break;
            case 2 : if((mutex == 1) && (full != 0))
            {
                consumer();
            }
        }
    }
}
```

```

    }
    else {
        printf("buffer is empty");
        break;
    }
}

case 3 : exit(0);
break;

}

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 item %d", x);
    mutex = signal(mutex);
}

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

```

outputs

1. producer
2. consumer
3. exit

enter your choice 1

~~producer~~
producer produces item 1
enter your choice 1

~~producer~~
producer produces item 2
enter your choice 2
enter your choice 2
consumer consumes item 2
~~enter your choice 2~~
enter your choice 2
consumer consumes item 1

~~enter your choice 2~~
enter your choice 2
~~buffer is empty~~
enter your choice 2
buffer is empty
~~enter your choice 3~~
enter your choice 3

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

Output

- 1. producer
- 2. consumer

3. exit

enter your choice 1

~~producer~~ producer produces item 1

enter your choice 1

~~producer~~

producer produces item 2

enter your choice 2

enter your choice 2

consumer consumes item 2

~~enter your choice 2~~

enter your choice 2

consumer consumes item 1

~~enter your choice~~

~~choice~~

enter your choice 2

buffer is empty

enter yo~~r~~

buffer is empty

~~enter your choice 3~~

enter your choice 3

8/16

3 Philosophers

```
return 0;

void simulate(int allow-two){
    int eating_count = 0;
    for(int i=0; i< num_hungry; ++i){
        int philosopher_id = hungry_philosophers[i];
        if(states[philosopher_id] == hungry){
            if(can_eat(philosopher_id)){
                states[philosopher_id] = eating;
                eating_count++;
                printf("P %d is granted to eat \n", philosopher_id+1);
                if(!allow_two && eating_count == 1)
                    break;
            }
            if(allow_two && eating_count == 2){
                break;
            }
        }
        for(int i=0; i< num_hungry; i++){
            int philosopher_id = hungry_philosophers[i];
            if(states[philosopher_id] == eating){
                int left_fork = philosopher_id;
                int rig
                int left_fork = philosopher_id;
                int right_fork = (philosopher_id + 1) % num_philosophers;
            }
        }
    }
}
```

```

forks[left_fork], forks[right_fork] = 0;
states[philosopher_id] = thinking;

int main(){
    printf(" enter the total number of philosophers (max %d)", max_philosophers);
    scanf("%d", &num_philosophers);
    if(num_philosophers < 2 || num_philosophers > max_philosophers){
        printf("invalid number");
        printf("invalid number of philosophers exiting\n");
        return 1;
    }
    printf(" how many are hungry\n");
    printf(" how many are hungry");
    scanf("%d", &num_hungry);
    for(int i=0; i<num_hungry; i++){
        printf(" enter philosopher %d position",
        printf(" enter philosopher %d position", i+1);
        int position;
        scanf("%d", &position);
        hungry_philosophers[i] = position-1;
        states[hungry_philosophers[i]] = hungry;
    }
    for(int i=0; i<num_philosophers; i++)
        forks[i] = 0;
}

```

```

int choice;
do{
    print_state();
    printf("\n 1. one can eat at a time\n 2 two can eat at a time\n 3 exit\n");
    printf("enter your choice");
    printf(" enter your choice");
    scanf("%d", &choice);
    switch(choice){
        case 1 : simulate();
        break;
        case 2 : simulate();
        break;
        case 3 : printf("exiting\n");
        break;
        default: printf("invalid choice\n");
        break;
    }
} while(choice != 3);

```

Output

```

enter total number of philosophers
how many are hungry : 2
enter philosopher 1 position 1
enter philosopher 2 position 1
1. one can eat at a time
2. two can eat at a time
3. exit
enter your choice 1
P1 is granted to eat

```

```

int choice;
do{
    print-state();
    printf("\n 1 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(" enter your choice");
    scanf("%d", &choice);
    switch(choice){
        case 1 : simulate(0);
        break;
        case 2 : simulate(1);
        break;
        case 3 : printf("exiting\n");
        break;
        default: printf("invalid choice please try again\n");
        break;
    }
}while(choice!=3);

OUTPUT
Enter total number 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
P1 is granted to eat

```

P₄ is granted to eat

- 1. one can eat at a time
- 2. two can eat at a time
- 3. exit

enter your choice : 2

P₁ and P₄ are granted

P₁ and P₄ are granted to eat

- 1. one can eat at a time
- 2. two can eat at a time
- 3. exit

enter your choice : 3

7) write a c program to simulate bankers algorithm for the purpose of deadlock avoidance

#include<st

#include<stdio.h>

int P, Y;

void avai(int all[P][Y], int tot[Y], int avail[Y]) {

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

avail[j] = 0;

}

for (int k = 0; k < P; k++) {

avail[j] =

avail[j] + all[k][j];

}

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

}

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

avail[j] = tot[j] - avail[j];

}

y
void needt(int all[P][Y], int max)

void needl(int all[P][Y], int max)

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

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

needmat[i][j] = max

}

}

y
void safety(int P, int Y, int all[P]

int f[P], C, count = 0, h = 0;

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

f[i] = 0;

}

while (count < P && h < P) {

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

if (f[i] == 0) {

C = 0;

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

for (int k = 0; k < P; k++) {

if (needmat

C = C + 1;

}

if (C == Y) {

printf("P%d is

for (int k = 0; k < P; k++) {

avail[k] +=

avail[k] +=

```

void need(int all[P][R], int max[R][C], -)
void need(int all[P][R], int max[R][C], int needmat[R][C]){
    for(int i=0; i<P; i++){
        for(int j=0; j<R; j++){
            needmat[i][j] = max[i][j] - all[i][j];
        }
    }
}

void safety(int P, int R, int all[P][R], int avail[R], int needmat[R][C], int seq[R]){
    int f[P], c, count=0, h=0;
    for(int i=0; i<P; i++){
        f[i]=0;
    }
    while(count<P && h<R){
        for(int i=0; i<P; i++){
            if(f[i]==0){
                C=0;
                for(j=0; j<R; j++){
                    if(all[i][j]>needmat[i][j] && avail[j]>0){
                        C=C+1;
                    }
                }
                if(C==R){
                    printf("%d is visited\n", i);
                    for(int k=0; k<R; k++){
                        avail[k] += needmat[i][k];
                        avail[k] -= all[i][k];
                    }
                    count++;
                }
            }
        }
    }
}

```

```

1     printf("%d", p);
2     printf("%d, %d)\n");
3     printf("%d\n");
4     printf("%d", avail[k]));
5     printf("\t");
6 }
7 printf("\n");
8 F[i]=1;
9 Count=Count+1;
10 seq[h]=i;
11 h=h+1;
12 }
13 }
14 }

15 int main()
16 {
17     printf("enter the number of processes");
18     scanf("%d",&P);
19     printf("enter the number of resources");
20     Scanf("%d",&R);
21     int tot[R], needmat[P][R], avail[R], seg[R];
22     printf("enter the total instances of each resource");
23     for(int i=0; i<R; i++)
24     {
25         for(int j=0; j<P; j++)
26         {
27             scanf("%d", &tot[i]);
28             scanf("%d", &needmat[j][i]);
29             scanf("%d", &avail[i]);
30         }
31     }
32     int all[P][R], max[R][P];
33     for(int i=0; i<P; i++)
34     {
35         for(int j=0; j<R; j++)
36         {
37             all[i][j] = max[i][j] = tot[i];
38         }
39     }
40     printf("enter details of each process");
41     for(int i=0; i<P; i++)
42     {
43         for(int j=0; j<R; j++)
44         {
45             scanf("%d", &all[i][j]);
46         }
47     }
48     printf("enter maximum matrix");
49     for(int i=0; i<P; i++)
50     {
51         for(int j=0; j<R; j++)
52         {
53             scanf("%d", &max[i][j]);
54         }
55     }
56     overall(all, tot, avail);
57     avail(all, tot, avail);
58     need(all, max, needmat);
59     need1(all, max, needmat);
60     printf("need matrix");
61     printf("need matrix\n");
62     for(int i=0; i<P; i++)
63     {
64         for(int j=0; j<R; j++)
65         {
66             printf("%d", needmat[i][j]);
67         }
68     }
69     printf("\n");
70
71     Safety(P, R, all, avail, needmat, seg);
72     printf("safe sequence is ");
73     printf("Safe sequence is ");
74     printf("Safe sequence is ");

```

```

    printf("enter details of each process (allocation matrix)\n");
    for(int i=0; i<P; i++){
        for(int j=0; j<R; j++){
            scanf("%d", &all[i][j]);
        }
    }

    printf("enter maximum matrix\n");
    for(int i=0; i<P; i++){
        for(int j=0; j<R; j++){
            scanf("%d", &max[i][j]);
        }
    }

    avail(all, tot, avail);
    avail(all, tot, avail);
    need(all, max, needmat);
    need1(all, max, needmat);

    printf("need mat");
    for(int i=0; i<P; i++){
        for(int j=0; j<R; j++){
            printf("%d", needmat[i][j]);
        }
    }
    printf("\n");
    Safety(P, R, all, avail, needmat, seq);
    printf("safe sequence is ");
    printf("%d", seq);
}

```

```

    if(i==0, i<p; i++)

for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        printf("P%d%d", seq[i]);
}

```

Output

enter the number of processes 5

enter the number of resources 3

enter available resources 3 3 2

enter maximum resources for each processes

3 2 2

9 0 2

2 2 2

4 3 3

enter the allocated proc

enter the allocated resources for each processes 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

Safety sequence P2 P3

Safety sequence P2 P3 P4 P5 P1

03/07/2024

8) write a c program to simulate deadlock detection

#include<

#include<stdio.h>

void main()

{

int n,m,i,j;

printf("enter the number of processes and number of types of resources\n");

\n");

```

scanf("%d %d", &n, &m);
scanf("%d %d", &un, &lm);
int max[n][m], need[n][m];
c = 0;
printf("enter the maximum
each process \n");

```

```

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

```

```

printf("enter the allocated
each process \n");

```

```

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

```

printf("enter the available resources\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] - alloc[i][j];
    }
}

```

```

scanf("%d %d", &n, &m);
scanf("%d %d", &n, &m);
int max[n][m], need[n][m], all[n][m], ava[m], flag=1, finish[n];
l = 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++)
    {
        for(j=0; j<m; j++)
        {
            if(finish[i]==0 && need[i][j]<=available[j])
            {
                c=c+1;
                if(c==m)
                {
                    for(j=0; j<m; j++)
                    {
                        available[j]=available[j]+alloc[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=j+1;
        flag=1;
    }
}
if(flag==1)
{
    printf("deadlock has occurred");
    printf("the deadlock processes are");
    printf("%d", dead[0]);
    for(i=1; i<j; i++)
    {
        printf(", %d", dead[i]);
    }
    else
    {
        printf("no deadlock has occurred");
    }
}

```

Output

enter the number of process
5
enter the maximum number each process

```

}
j=0;
flag=0;
for(i=0; i<n; i++)
{
    if(finish[i]==0)
    {
        dead[j]=i;
        j=j+1;
        flag=1;
    }
}
if(flag==1)
{
    printf(" deadlock has occurred \n");
    printf(" the deadlock processes are ");
    printf(" the deadlock processes are \n");
    for(i=0; i<n; i++)
    {
        printf("%d", dead[i]);
    }
}
else
{
    printf(" no deadlock has occurred \n");
}

```

Output:

enter the number of process and number of types of resources
 5 3
 enter the maximum number of each type of resource needed by each process

7 5 3
3 2 2
9 0 2
2 2 4
4 3 3

~~enter the allocated number of each~~

enter the allocated number of each type of resource needed by each process

0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

enter the available number of each type of resource

3 3 2

~~deadlock~~

no deadlock has occurred

~~q) write a c program to simulate the follow~~

q) write a c program to simulate the following contiguous memory allocation techniques

- (a) worst-fit
- (b) best-fit
- ~~worst-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("m~~

```
'printf("memory man
printf("file.no")
printf(" file.no")
for(int i=0; i<n_file
```

```
for(int j=0; j<
```

```
if(blocks[j].is_f
```

```
blocks[j].is_f
```

```
'printf("memory man
```

```
'printf("file.no")
for(int i=0; i<n_file
```

```
int worst_fit_block
```

```
int max_fragment
```

```
for(int j=0; j<n-
```

```
if(blocks[j].is_f
```

```
int frag
```

```
if(fragm
```

```

    printf("memory management scheme - First fit\n");
    printf("file-no\t file-size\t block-no\t block-size\t fragment\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%dt\t%d\t%d\t", 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\t file-size\t block-no\t block-size\t fragment\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;
                }
            }
        }
    }
}

```

```

        worst-fit-block = j;
    }
}

if(worst-fit-block) == -1 {
    blocks[worst-fit-block].is_free = 0;
    printf("%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) {
    }
}

```

```

block[best-fit] = block

printf("%d\n", best-fit);
file[i].file-size;
blocks[best-fit];
blocks[best-fit];

}

int main()
{
    int n-blocks, n-files;
    printf("enter number of blocks");
    scanf("%d", &n-blocks);

    printf("enter 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;
        printf("enter size of block %d", i);
        scanf("%d", &blocks[i].size);
        blocks[i].is-free = 1;
    }

    struct file filest;
    struct file files[n-files];
    for(int i=0; i<n-files; i++)
    {
        printf("enter size of file %d", i);
        scanf("%d", &files[i].size);
        files[i].is-free = 1;
    }

    firstfit(blocks, n-blocks, files, n-files);
}

```

```

blocks[best-fit-block].is-free = 0;

printf("%d\t%d\t%d\t%d\t%d\n", files[i].file-no,
      file-size, blocks[best-fit-block].block-no,
      blocks[best-fit-block].file-size, min-fragment);

max-fragment);

}

int main()
{
    int n-blocks, n-files;
    printf("enter number of blocks ");
    scanf("%d", &n-blocks);
    printf("enter 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 size of block %d", i+1);
        blocks[i].is-free = 1;
    }

    struct file files;
    struct file files[n-files];
    for(int i=0; i<n-files; i++)
    {
        printf("enter 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);

```

V bestfit(blocks, n-blocks, files, n-files);

Output

enter the number of blocks 3
 enter the number of files 2
 enter the size of block1 5
 enter the size of block2 2
 enter the size of block3 7
 enter the size of file1 1
 enter the size of file2 4

memory management scheme - first fit

file no	file size	block no	block size	fragment
1	1	1	8	4
2	4	3	7	3

memory management scheme worst-fit

file no	file size	block no	block size	fragment
1	1	3	7	1
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

10/07/2024
 10) write a c program to simulate pag memory management

```

#include<stdio.h>
#include<climits.h>
#include<cslib.h>

```

void print_frames(int frame[], int capacity,

```
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(" PF No %d", page-faultb);
}
```

```
}
```

```
printf("\n");
```

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

```
int frame[capacity], index=0, page-fa
```

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

```
{
```

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

```
}
```

```
printf("FIFO page replacement process");
```

```
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;
```

```
}
```

10/03/2024
10) write a c program to simulate paging technique of memory management.

```
#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("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 Processes\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==0)
        {
            frame[index]=pages[i];
            index=(index+1)%capacity;
            page_faults++;
        }
    }
}
```

```

        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\n");
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;
    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, min_index;
            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);
}

void optimal(int pages[], int n, int capacity, page_faults){
    int frame[capacity], page_faults = 0;
    for(int i=0; i<capacity; i++)
        frame[i] = -1;
    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;
            for(int j=0; j<capacity; j++){
                if(frame[j] != -1 && pages[i] != frame[j])
                    farthest = j;
            }
            frame[farthest] = pages[i];
            page_faults++;
        }
    }
    print_frames(frame, capacity);
}

```

```

        min_index=j;
    }

    frame[min_index]=pages[i];
    counter[min_index]=time+i;
    page_faults++;

}

printf("print_frames(frame, capacity, found ? 0 : page_faults);");

printf("total page faults using LRU %d\n\n", page_faults);

};

faults=0;

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++){
                for(int k=0; 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++;
}
printf("frames(%d, capacity %d, found %d) : page-faults %d\n", 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(i=0; i < n; i++)
        scanf("%d", &pages[i]);
    printf("enter the frame capacity");
    scanf("%d", &capacity);
    printf("\n Pages");
}

```

```

for(int i=0; i < n; i++){
    printf("%d", pages[i]);
}
printf("\n");
fifo(pages, n, capacity);
lru(pages, n, capacity);
optimal(pages, n, capacity);
free(pages);
}

```

Output

```

enter the number of pages 20
enter the pages 7 0 1 2 0 3 0
enter the frame capacity 3
pages: 7 0 1 2 0 3 0 4 2 3 0
-----page
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
-----page
4 2 0
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

```

```

    for(int i=0; i<n; i++){
        printf("%d", pages[i]);
    }
    printf("\n");
    fifo(pages, n, capacity);
    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    optimal(pages, n, capacity);
    free
    free(pages);
}

```

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

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

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

~~4 3 0~~

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

Colour Me



lru page replacement process:

lru page replacement process:

7 - -	PF no 1	7 - -	PF no 1
0 - -	PF no 2	0 - -	PF no 2
0 1 -	PF no 3	0 1 -	PF no 3
0 1 2	PF no 4	0 1 2	PF no 4
0 3 2	PF no 5	0 1 2	PF no 5
0 3 2	PF no 5	0 3 2	PF no 6
0 3 4	PF no 6	0 3 4	PF no 6
0 2 4	PF no 7	0 2 4	PF no 7
3 2 4	PF no 8	3 2 4	PF no 8
3 2 0	PF no 9	3 2 0	PF no 9
3 2 1	PF no 10	3 2 1	PF no 10
0 2 1	PF no 10	3 2 1	PF no 10
0 2 1	PF no 11	0 2 1	PF no 11
0 7 1	PF no 11	0 2 1	PF no 11
0 7 1	PF no 12	0 7 1	PF no 12
0 7 1	PF no 12	0 7 1	PF no 12

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	PF no 5
2 0 3	PF nos 5 PF no 5
2 0 3	PF no 6
2 4 3	PF no 6
2 4 3	PF no 7
2 0 3	PF no 7
2 0 3	PF no 8
2 0 1	PF no 8
2 0 1	PF no 9
7 0 1	PF no 9
7 0 1	PF no 9

Total page faults op'l

Total page faults using optimal 9