

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT on OPERATING SYSTEMS

Submitted by

Trishul (1WA23CS023)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Feb-2025 to June-2025

B. M. S. College of Engineering,  
Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

**CERTIFICATE**

This is to certify that the Lab work entitled “OPERATING SYSTEMS 23CS4PCOPS” carried out by Trishul (1WA23CS023), who is Bonafide student of B. M. S. College of Engineering. It is inpartial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025-June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS - (23CS4PCOPS) work prescribed for the said degree.

Ms Seema Patil  
Assistant Professor  
Department of CSE  
BMSCE, Bengaluru

Dr. Kavitha Sooda  
Professor and Head  
Department of CSE  
BMSCE, Bengaluru

## Index Sheet

Sl. No.	Experiment Title	Page No.
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)	1
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)	17
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.	30
4	Write a C program to simulate Real-Time CPU Scheduling algorithms: a) Rate- Monotonic b) Earliest-deadline First c) Proportional scheduling	36
5	Write a C program to simulate producer-consumer problem using semaphores	48
6	Write a C program to simulate the concept of Dining Philosophers problem.	53
7	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.	59
8	Write a C program to simulate deadlock detection	64
9	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	69
10	Write a C program to simulate page replacement algorithms a)FIFO b) LRU c) Optimal	78

C01	Apply the different concepts and functionalities of Operating System
C02	Analyse various Operating system strategies and techniques
C03	Demonstrate the different functionalities of Operating System.
C04	Conduct practical experiments to implement the functionalities of Operating system.

Course outcome

## Program 1

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

a) First Come First Serve

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

typedef struct {
    int id, arrival, burst, remaining, waiting, turnaround, completion, response, started;
} Process;

void swap(Process *a, Process *b) {
    Process temp = *a;
    *a = *b;
    *b = temp;
}

void sortByArrival(Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].arrival > p[j + 1].arrival) {
                swap(&p[j], &p[j + 1]);
            }
        }
    }
}
```

```

void fcfs(Process p[], int n) {
    sortByArrival(p, n);
    int time = 0;

    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival)
            time = p[i].arrival;

        p[i].response = time - p[i].arrival;
        p[i].completion = time + p[i].burst;
        p[i].turnaround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        time = p[i].completion;
    }
}

void displayResults(Process p[], int n, const char *title) {
    printf("\n--- %s ---\n", title);
    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\tRT\n");

    float totalWT = 0, totalTAT = 0, totalRT = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].arrival, p[i].burst,
               p[i].completion, p[i].turnaround, p[i].waiting, p[i].response);

        totalWT += p[i].waiting;
        totalTAT += p[i].turnaround;
        totalRT += p[i].response;
    }
}

```

```

printf("Average Waiting Time: %.2f\n", totalWT / n);
printf("Average Turnaround Time: %.2f\n", totalTAT / n);
printf("Average Response Time: %.2f\n", totalRT / n);

}

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

    Process p[n];

    printf("Enter Arrival Time and Burst Time:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        scanf("%d %d", &p[i].arrival, &p[i].burst);
        p[i].remaining = p[i].burst;
        p[i].waiting = p[i].turnaround = p[i].completion = p[i].response = p[i].started = 0;
    }

    fcfs(p, n);
    displayResults(p, n, "First Come First Serve (FCFS)");

    return 0;
}

```

}O/P:

```

Enter number of processes: 4
Enter Arrival Time and Burst Time for each process:
P[1]: 0
7
P[2]: 0
3
P[3]: 0
4
P[4]: 0
6
(FCFS)
PID Arrival Burst Completion Turnaround Waiting
1      0      7      7      7      0
2      0      3      10     10     7
3      0      4      14     14     10
4      0      6      20     20     14

Average Turnaround Time: 12.75
Average Waiting Time: 7.75

```

Teacher's Sign

PAGE NO: \_\_\_\_\_  
DATE: \_\_\_\_\_

LAB-1  
Write a program to simulate the following non-preemptive CPU scheduling algorithm to find turnaround time & waiting time.

1] FCFS  
2] SJF (non-preemptive & preemptive)

1) → #include <stdio.h>

typedef struct {  
 int id, arrival, burst, completion, turnaround,  
 float waiting;  
} Process;

void sortByArrival(Process P[], int n){  
 for(int i=0; i<n-1; i++)  
 for(int j=0; j<n-i-1; j++)  
 if(P[j].arrival > P[j+1].arrival){  
 Process temp = P[i];  
 P[i] = P[j+1];  
 P[j+1] = temp;  
 }  
}

void fcfs(Process P[], int n, float \*avgTAT, float \*avgWT){  
 sortByArrival(P, n);  
 int time = 0, totalTAT = 0, totalWT = 0;  
 for(int i=0; i<n; i++){  
 if(time < P[i].arrival)  
 time = P[i].arrival;  
 P[i].completion = time + P[i].burst;  
 totalTAT += P[i].completion - P[i].arrival;  
 totalWT += P[i].completion - P[i].arrival - P[i].burst;  
 }  
 \*avgTAT = totalTAT / n;  
 \*avgWT = totalWT / n;  
}

$P[i].turnaround = P[i].completion - P[i].arrival$   
 $P[i].waiting = P[i].burst - P[i].turnaround$   
 $avgTAT = \frac{1}{n} \sum P[i].turnaround$   
 $avgWT = \frac{1}{n} \sum P[i].waiting$

```

void display(Process p[], int n, float avgTAT,
            float avgWT) {
    printf(" PID Arrival Burst Completion Turnaround
           Waiting\n");
    for (int i=0; i<n; i++) {
        printf("%d %d %d %d %d %d %d
               %d", p[i].id, p[i].arrival, p[i].burst, p[i].completion,
               p[i].turnaround, p[i].waiting);
    }
    printf("\n");
    printf("In Average turnaround time : %.2f", avgTAT);
    printf("In Average waiting time : %.2f", avgWT);
}

int main() {
    int n;
    float avgTAT, avgWT;
    printf("Enter no. of processes : ");
    scanf("%d", &n);
    Process p[n];
    printf("Enter Arrival Time & Burst Time ");
    for (int i=0; i<n; i++) {
        p[i].arrival = i+1;
        p[i].burst = rand() % 10 + 1;
    }
    display(p, n, avgTAT, avgWT);
}

```

PID	Arrival	Burst	Completion	Turnaround	Waiting
1	0	2	2	2	0
2	0	3	5	5	2
3	0	4	9	9	9
4	0	6	15	15	15

Average turnaround time = 12.75  
 Average waiting time = 7.75

## Program 1

b) Shortest Job First (non preemptive)

Programme:

```
#include <stdio.h>
#include <limits.h>
typedef struct {
    int id, arrival, burst, completion, turnaround, waiting;
} Process;

void sortByArrival(Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].arrival > p[j + 1].arrival) {
                Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

// SJF Scheduling (Non-Preemptive)
void sjf_non_preemptive(Process p[], int n, float *avgTAT, float *avgWT) {
    int completed = 0, time = 0, minIdx, totalTAT = 0, totalWT = 0;
    int isCompleted[n];
    for (int i = 0; i < n; i++) isCompleted[i] = 0;
```

```

while (completed < n) {
    minIdx = -1;
    int minBurst = INT_MAX;
    for (int i = 0; i < n; i++) {
        if (!isCompleted[i] && p[i].arrival <= time && p[i].burst < minBurst) {
            minBurst = p[i].burst;
            minIdx = i;
        }
    }
    if (minIdx == -1) { time++; continue; }

    p[minIdx].completion = time + p[minIdx].burst;
    p[minIdx].turnaround = p[minIdx].completion - p[minIdx].arrival;
    p[minIdx].waiting = p[minIdx].turnaround - p[minIdx].burst;
    time = p[minIdx].completion;
    isCompleted[minIdx] = 1;
    totalTAT += p[minIdx].turnaround;
    totalWT += p[minIdx].waiting;
    completed++;
}

*avgTAT = (float)totalTAT / n;
*avgWT = (float)totalWT / n;
}

void display(Process p[], int n, float avgTAT, float avgWT) {
    printf("\nPID Arrival Burst Completion Turnaround Waiting\n");
    for (int i = 0; i < n; i++) {
        printf("%3d %7d %6d %10d %10d %8d\n", p[i].id, p[i].arrival, p[i].burst,
p[i].completion, p[i].turnaround, p[i].waiting);
    }
}

```

```

    }

printf("\nAverage Turnaround Time: %.2f", avgTAT);
printf("\nAverage Waiting Time: %.2f\n", avgWT);

}

int main() {
    int n;
    float avgTAT, avgWT;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("P[%d]: ", i + 1);
        scanf("%d %d", &p[i].arrival, &p[i].burst);
    }

    printf("\nShortest Job First (Non-Preemptive) Scheduling\n");
    sjf_non_preemptive(p, n, &avgTAT, &avgWT);
    display(p, n, avgTAT, avgWT);

    return 0;
}

```

O/p:

Enter number of processes: 4

Enter Arrival Time and Burst Time for each process:

P[1]: 0

7

P[2]: 0

3

P[3]: 0

4

P[4]: 0

6

### Shortest Job First (Non-Preemptive) Scheduling

PID	Arrival	Burst	Completion	Turnaround	Waiting
1	0	7	20	20	13
2	0	3	3	3	0
3	0	4	7	7	3
4	0	6	13	13	7

Average Turnaround Time: 10.75

Average Waiting Time: 5.75

Q. SJF (non-preemptive)  
#include <limits.h>

```
void sjf_non_preemptive (Process p[], int n, float avgTAT, float avgWT)
{
    int completed = 0, tIme = 0, minIdx, totalTAT = 0,
        totalWT = 0;
    int isCompleted[n];
    for (int i=0; i<n; i++) isCompleted[i] = 0;

    while (completed < n) {
        minIdx = -1;
        int minBurst = INTMAX;
        for (int i=0; i<n; i++) {
            if (!isCompleted[i] && p[i].arrival <= tIme && p[i].burst < minBurst) {
                minBurst = p[i].burst;
                minIdx = i;
            }
        }

        if (minIdx == -1) {
            tIme++;
            continue;
        }

        p[minIdx].completion = tIme + p[minIdx].burst;
        p[minIdx].turnaround = p[minIdx].completion -
            p[minIdx].arrival;
        tIme = p[minIdx].completion;
        isCompleted[minIdx] = 1;
        completed++;
    }
}
```

$\text{P}[\text{minIdx}].\text{completion} = \text{tIme} + \text{p}[\text{minIdx}].\text{burst}$   
 $\text{P}[\text{minIdx}].\text{turnaround} = \text{p}[\text{minIdx}].\text{completion} - \text{p}[\text{minIdx}].\text{arrival}$   
 $\text{tIme} = \text{p}[\text{minIdx}].\text{completion}$   
 $\text{isCompleted}[\text{minIdx}] = 1$

PAGE NO :  
DATE :

```

totalTAT += p[minIdx].turnaround;
totalWT += p[minIdx].waiting;
completed += 1;
}

avgTAT = (float) totalTAT / n;
avgWT = (float) totalWT / n;

int main() {
    int n;
    float avgTAT, avgWT;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process P[n];
    printf("Enter Arrival & Burst time for each process");
    for(int i=0; i<n; i++) {
        P[i].id = i+1;
        printf("P[%d].id = %d\n", i, P[i].id);
        scanf("%d", &P[i].arrival);
        scanf("%d", &P[i].burst);
    }
    sort("id -> id", &P[0].arrival, &P[n-1].burst);
    printf("SJF");
    SJF_non_preeemptive(P, n, avgTAT, avgWT);
    display(P, n, avgTAT, avgWT);
    return 0;
}

```

~~o/p~~

~~P[0] = 0~~  
~~P[1] = 0~~  
~~P[2] = 0~~  
~~P[3] = 0~~

ID	Arrival	Burst	Completion	Turnaround	Waiting
1	0	7	20	20	13
2	0	3	13	13	0
3	0	4	7	7	0
4	0	6	13	13	7

Average Turnaround Time = 10.75  
Average Waiting Time = 5.75

P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>1</sub>	Waiting
Arrived at 0	3	7	13	20

(10.75)

## Program 1

c) Shortest Job First (preemptive)

Programme:

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

typedef struct {
    int id, arrival, burst, remaining, completion, turnaround, waiting;
} Process;

void sortByArrival(Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].arrival > p[j + 1].arrival) {
                Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

void sjf_preemptive(Process p[], int n, float *avgTAT, float *avgWT) {
    int completed = 0, time = 0, minIdx, totalTAT = 0, totalWT = 0;
    int isCompleted[n];
    for (int i = 0; i < n; i++) {
        isCompleted[i] = 0;
        p[i].remaining = p[i].burst;
    }
}
```

```
}
```

```
while (completed < n) {  
    minIdx = -1;  
    int minBurst = INT_MAX;  
    for (int i = 0; i < n; i++) {  
        if (!isCompleted[i] && p[i].arrival <= time && p[i].remaining < minBurst &&  
            p[i].remaining > 0) {  
            minBurst = p[i].remaining;  
            minIdx = i;  
        }  
    }  
    if (minIdx == -1) { time++; continue; }  
  
    p[minIdx].remaining--;  
    time++;  
    if (p[minIdx].remaining == 0) {  
        p[minIdx].completion = time;  
        p[minIdx].turnaround = p[minIdx].completion - p[minIdx].arrival;  
        p[minIdx].waiting = p[minIdx].turnaround - p[minIdx].burst;  
        isCompleted[minIdx] = 1;  
        totalTAT += p[minIdx].turnaround;  
        totalWT += p[minIdx].waiting;  
        completed++;  
    }  
}  
*avgTAT = (float)totalTAT / n;  
*avgWT = (float)totalWT / n;  
}
```

```

void display(Process p[], int n, float avgTAT, float avgWT) {
    printf("\nPID Arrival Burst Completion Turnaround Waiting\n");
    for (int i = 0; i < n; i++) {
        printf("%3d %7d %6d %10d %10d %8d\n", p[i].id, p[i].arrival, p[i].burst,
               p[i].completion, p[i].turnaround, p[i].waiting);
    }
    printf("\nAverage Turnaround Time: %.2f", avgTAT);
    printf("\nAverage Waiting Time: %.2f\n", avgWT);
}

int main() {
    int n;
    float avgTAT, avgWT;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("P[%d]: ", i + 1);
        scanf("%d %d", &p[i].arrival, &p[i].burst);
    }

    printf("\nShortest Job First (Preemptive) Scheduling\n");
    sjf_preemptive(p, n, &avgTAT, &avgWT);
    display(p, n, avgTAT, avgWT);
}

```

```
    return 0;  
}  
  
O/P:
```

- PS C:\Users\trish\OneDrive\Desktop\OS\_LAB> gcc -o sjf2 sjf2.c
- PS C:\Users\trish\OneDrive\Desktop\OS\_LAB> ./sjf2.exe  
Enter number of processes: 4  
Enter Arrival Time and Burst Time for each process:  
P[1]: 0  
2  
P[2]: 1  
6  
P[3]: 2  
8  
P[4]: 3  
4

#### Shortest Job First (Preemptive) Scheduling

PID	Arrival	Burst	Completion	Turnaround	Waiting
1	0	2	2	2	0
2	1	6	12	11	5
3	2	8	20	18	10
4	3	4	7	4	0

Average Turnaround Time: 8.75

Average Waiting Time: 3.75

SJT (PREEMPTIVE) ~~for(j=0; j<n; j++)~~  
Date: 01/03/2023

```

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

typedef struct {
    int id, arrival, burst, remaining, completion,
        turnaround, waiting;
} process;

```

Void sortByArrival(Process p[], int n)

```

for(int i=0; i<n-1; i++) {
    for(int j=i+1; j<n-i-1; j++) {
        if(p[i].arrival > p[j].arrival) {
            Process temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    }
}

```

PC<sub>j+1</sub> = item p<sub>j</sub>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	8010	8011	8012	8013	8014	8015	8016	8017	8018	8019	8020	8021	8022	8023	8024	8025	8026	8027	8028	8029	8030	8031	8032	8033	8034	8035	8036	8037	8038	8039	8040	8041	8042	8043	8044	8045	8046	8047	8048	8049	8050	8051	8052	8053	8054	8055	8056	8057	8058	8059	8060	8061	8062	8063	8064	8065	8066	8067	8068	8069	8070	8071	8072	8073	8074	8075	8076	8077	8078	8079	8080	8081	8082	8083	8084	8085	8086	8087	8088	8089	8090	8091	8092	8093	8094	8095	8096	8097	8098	8099	80100	80101	80102	80103	80104	80105	80106	80107	80108	80109	80110	80111	80112	80113	80114	80115	80116	80117	80118	80119	80120	80121	80122	80123	80124	80125	80126	80127	80128	80129	80130	80131	80132	80133	80134	80135	80136	80137	80138	80139	80140	80141	80142	80143	80144	80145	80146	80147	80148	80149	80150	80151	80152	80153	80154	80155	80156	80157	80158	80159	80160	80161	80162	80163	80164	80165	80166	80167	80168	80169	80170	80171	80172	80173	80174	80175	80176	80177	80178	80179	80180	80181	80182	80183	80184	80185	80186	80187	80188	80189	80190	80191	80192	80193	80194	80195	80196	80197	80198	80199	80200	80201	80202	80203	80204	80205	80206	80207	80208	80209	80210	80211	80212	80213	80214	80215	80216	80217	80218	80219	80220	80221	80222	80223	80224	80225	80226	80227	80228	80229	80230	80231	80232	80233	80234	80235	80236	80237	80238	80239	80240	80241	80242	80243	80244	80245	80246	80247	80248	80249	80250	80251	80252	80253	80254	80255	80256	80257	80258	80259	80260	80261	80262	80263	80264	80265	80266	80267	80268	80269	80270	80271	80272	80273	80274	80275	80276	80277	80278	80279	80280	80281	80282	80283	80284	80285	80286	80287	80288	80289	80290	80291	80292	80293	80294	80295	80296	80297	80298	80299	80300	80301	80302	80303	80304	80305	80306	80307	80308	80309	80310	80311	80312	80313	80314	80315	80316	80317	80318	80319	80320	80321	80322	80323	80324	80325	80326	80327	80328	80329	80330	80331	80332	80333	80334	80335	80336	80337	80338	80339	80340	80341	80342	80343	80344	80345	80346	80347	80348	80349	80350	80351	80352	80353	80354	80355	80356	80357	80358	80359	80360	80361	80362	80363	80364	80365	80366	80367	80368	80369	80370	80371	80372	80373	80374	80375	80376	80377	80378	80379	80380	80381	80382	80383	80384	80385	80386	80387	80388	80389	80390	80391	80392	80393	80394	80395	80396	80397	80398	80399	80400	80401	80402	80403	80404	80405	80406	80407	80408	80409	80410	80411	80412	80413	80414	80415	80416	80417	80418	80419	80420	80421	80422	80423	80424	80425	80426	80427	80428	80429	80430	80431	80432	80433	80434	80435	80436	80437	80438	80439	80440	80441	80442	80443	80444	80445	80446	80447	80448	80449	80450	80451	80452	80453	80454	80455	80456	80457	80458	80459	80460	80461	80462	80463	80464</td
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-----------

```

if (p[minIdx].remaining == 0)
    p[minIdx].completion = time;
p[minIdx].turnaround = p[minIdx].completion - p[minIdx].arrival;
p[minIdx].waiting = p[minIdx].turnaround - p[minIdx].burst;
isCompleted(p[minIdx]);
totalTAT += p[minIdx].turnaround;
totalWT += p[minIdx].waiting;
completed++;
3
avgTAT = (float) totalTAT / n;
avgWT = (float) totalWT / n;
3
void display (Process p[], int n, float avgTAT, float avgWT)
{
    printf ("In Avg TAT %f\n", avgTAT);
    printf ("In Average WT %f\n", avgWT);
}
int main()
{
    int n;
    float avgTAT, avgWT;
}

```

PAGE NO: \_\_\_\_\_  
DATE: \_\_\_\_\_

```

printf ("Enter no of processes");
scanf ("%d", &n);
Process p[n];
printf ("Enter Arrival time & Burst time");
for (int i=0; i<n; i++)
{
    p[i].id = i+1;
    printf ("P[%d]: ", i+1);
    scanf ("%d-%d", &p[i].arrival, &p[i].burst);
}
3
printf ("In SPT");
SJF(p, n, avgTAT, avgWT);
display (p, n, avgTAT, avgWT);
return 0;
3

```

## Program 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)

Programme:

```
#include <stdio.h>

#define MAX 100

void priorityPreemptive(int n, int at[], int bt[], int pr[]) {
    int ct[n], tat[n], wt[n], rem_bt[n], is_completed[n];
    int time = 0, completed = 0, min_priority, index;

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

    while (completed < n) {
        min_priority = 9999;
        index = -1;

        for (int i = 0; i < n; i++) {
            if (at[i] <= time && is_completed[i] == 0 && pr[i] < min_priority && rem_bt[i] > 0)
            {
                min_priority = pr[i];
                index = i;
            }
        }

        if (index != -1) {
            time += rem_bt[index];
            ct[index] = time;
            rem_bt[index] = 0;
            is_completed[index] = 1;
            completed++;
        }
    }
}
```

```

if (index == -1) {
    time++;
} else {
    rem_bt[index]--;
    time++;

    if (rem_bt[index] == 0) {
        ct[index] = time;
        is_completed[index] = 1;
        completed++;
    }
}

float total_tat = 0, total_wt = 0;
printf("\nP#\tAT\tBT\tPR\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    total_tat += tat[i];
    total_wt += wt[i];
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], pr[i], ct[i], tat[i], wt[i]);
}

printf("Average TAT: %.2f\n", total_tat / n);
printf("Average WT: %.2f\n", total_wt / n);
}

```

```

void priorityNonPreemptive(int n, int at[], int bt[], int pr[]) {
    int ct[n], tat[n], wt[n], is_completed[n], rem_bt[n];
    int time = 0, completed = 0;

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

    while (completed < n) {
        int min_priority = 9999, index = -1;

        for (int i = 0; i < n; i++) {
            if (at[i] <= time && is_completed[i] == 0 && pr[i] < min_priority) {
                min_priority = pr[i];
                index = i;
            }
        }

        if (index == -1) {
            time++;
        } else {
            time += bt[index];
            ct[index] = time;
            is_completed[index] = 1;
            completed++;
        }
    }
}

```

```

float total_tat = 0, total_wt = 0;
printf("\nP#\tAT\tBT\tPR\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    total_tat += tat[i];
    total_wt += wt[i];
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], pr[i], ct[i], tat[i], wt[i]);
}

printf("Average TAT: %.2f\n", total_tat / n);
printf("Average WT: %.2f\n", total_wt / n);
}

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

    int at[n], bt[n], pr[n];
    for (int i = 0; i < n; i++) {
        printf("Enter AT, BT, and Priority P%d: ", i + 1);
        scanf("%d %d %d", &at[i], &bt[i], &pr[i]);
    }

    printf("\nChoose Scheduling Algorithm:\n");
    printf("1. Preemptive Priority Scheduling\n");
    printf("2. Non-Preemptive Priority Scheduling\n");
    printf("Enter choice: ");
}

```

```

scanf("%d", &choice);

if (choice == 1) {
    priorityPreemptive(n, at, bt, pr);
} else if (choice == 2) {
    priorityNonPreemptive(n, at, bt, pr);
} else {
    printf("Invalid choice!\n");
}

return 0;
}

```

O/P:

```

PS C:\Users\trish\OneDrive\Desktop\OS_LAB> gcc priority.c -o priority
PS C:\Users\trish\OneDrive\Desktop\OS_LAB> ./priority
Enter number of processes: 5
Enter AT, BT, and Priority P1: 0
10
3
Enter AT, BT, and Priority P2: 0
1
1
Enter AT, BT, and Priority P3: 3
2
3
Enter AT, BT, and Priority P4: 5
1
4
Enter AT, BT, and Priority P5: 10
5
2

Choose Scheduling Algorithm:
1. Preemptive Priority Scheduling
2. Non-Preemptive Priority Scheduling
Enter choice: 2

P#      AT       BT       PR       CT       TAT      WT
1        0        10       3        11       11       1
2        0        1        1        1        1        0
3        3        2        3        18       15       13
4        5        1        4        19       14       13
5       10        5        2        16       6        1

Average TAT: 9.40
Average WT: 5.60

```

i) Priority (Preemptive).  
 includes <stdio.h>  
 includes <limits.h>  
 defines struct  
 int id, arrival, burst, remaining, priority,  
 computation, turnaround, waiting;  
 3 process;  
 void priority (Process P[], int n, float avgTAT,  
 float avgWT);  
 int completed = 0, time = 0, minIdt, turnaround  
 totalWT = 0;  
 int isCompleted(i);  
 for (int i=0; i<n; i++)  
 if (completed[i] == 0)  
 P[i].remaining = P[i].burst;  
 while (completed < n) {  
 minIdt = -1;  
 int minPriority = INT\_MAX;  
 for (int i=0; i<n; i++)  
 if (!isCompleted[i] && P[i].arrival <= time && P[i].remaining > 0)  
 if (P[i].priority < minPriority ||  
 (P[i].priority == minPriority &&  
 P[i].arrival < P[minIdt].arrival))  
 minPriority = P[i].priority;  
 minIdt = i;  
 if (minIdt == -1)  
 time++;  
 P[minIdt].remaining--;  
 time++;  
 if (P[minIdt].remaining == 0){  
 P[minIdt].computationTime =  
 P[minIdt].turnaround = P[minIdt].computation -  
 P[minIdt].arrival;  
 P[minIdt].waiting = P[minIdt].turnaround -  
 P[minIdt].burst;  
 if (completed[minIdt] == 0)  
 completed[minIdt] = 1;  
 totalTAT += P[minIdt].turnaround;  
 totalWT += P[minIdt].waiting;  
 completed++;  
 }  
 }  
 avgTAT = (float)totalTAT / n;  
 avgWT = (float)totalWT / n;  
 }

void display (Process P[], int n, float avgTAT,  
 float avgWT);  
 printf (" Arrival Burst Priority Completion  
 turnaround waiting ");  
 for (int i=0; i<n; i++) {  
 }

PAGE NO.:  
 DATE:

point P(10, 3d) - 17d - 16d - 18d - 110d - 10d  
 - 18d",  
 P[1].id, P[1].arrival, P[1].burst, P[1].time  
 P[1].completion - P[1].turnaround, P[1].waiting

3  
 points (n, in Average Turnaround Time : -1.4f", avgTAT)  
 point P(n, in Average Waiting Time : -1.2f", avgW)

3  
 i.e. main C.R.  
 i.e. n;  
 float avgTAT, avgWT;

point P(n, 0) & process P;  
 start ("1.d", Pn);

Priority r(n);  
 priority P(Ende A.T., B.T., Priority Two each process)

for (i = 1; i <= n; i++) {  
 point P(P[i].id = i + 1), P[i].arrival  
 start ("1.d", "1.d", &P[i].arrival, &P[i].burst,  
 &P[i].priority);

3  
 priority ("Priority scheduling");

priority (P, n, AvgTAT, AvgWT),  
 Disp (P, n, AvgTAT, AvgWT)

action O;  
 3

End no & processes = 4  
 End no Arrival Time, Burst Time, Priority:  
 P[1] = 0  
 10 3

PAGE NO.:  
 DATE:

P[2] = 0 1 1  
 P[3] = 3 2 3  
 P[4] = 5 1 4

Preemptive Priority scheduling

	PID	Arrival	Burst	Priority	Completion	TAT	WT
1	0	10	3	1	11	11	1
2	0	11	1	1	12	1	0
3	3	12	3	1	15	10	8
4	4	14	4	1	19	9	8

Avg TAT AM: 12.25  
 Avg WT : 4.25

✓ C.P.W. [19] = min. waiting time [19]

## Program 2

Round Robin

Program:

```
#include <stdio.h>

#define MAX 100

void roundRobin(int n, int at[], int bt[], int quant) {
    int ct[n], tat[n], wt[n], rem_bt[n];
    int queue[MAX], front = 0, rear = 0;
    int time = 0, completed = 0, visited[n];

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

    queue[rear++] = 0;
    visited[0] = 1;

    while (completed < n) {
        int index = queue[front++];

        if (rem_bt[index] > quant) {
            time += quant;
            rem_bt[index] -= quant;
        } else {
            time += rem_bt[index];
            completed++;
        }
    }
}
```

```

rem_bt[index] = 0;
ct[index] = time;
completed++;
}

for (int i = 0; i < n; i++) {
    if (at[i] <= time && rem_bt[i] > 0 && !visited[i]) {
        queue[rear++] = i;
        visited[i] = 1;
    }
}

if (rem_bt[index] > 0) {
    queue[rear++] = index;
}

if (front == rear) {
    for (int i = 0; i < n; i++) {
        if (rem_bt[i] > 0) {
            queue[rear++] = i;
            visited[i] = 1;
            break;
        }
    }
}

float total_tat = 0, total_wt = 0;
printf("P#\tAT\tBT\tCT\tTAT\tWT\n");

```

```

for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    total_tat += tat[i];
    total_wt += wt[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("Average TAT: %.2f\n", total_tat / n);
printf("Average WT: %.2f\n", total_wt / n);
}

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

    int at[n], bt[n];
    for (int i = 0; i < n; i++) {
        printf("Enter AT and BT for process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
    }

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

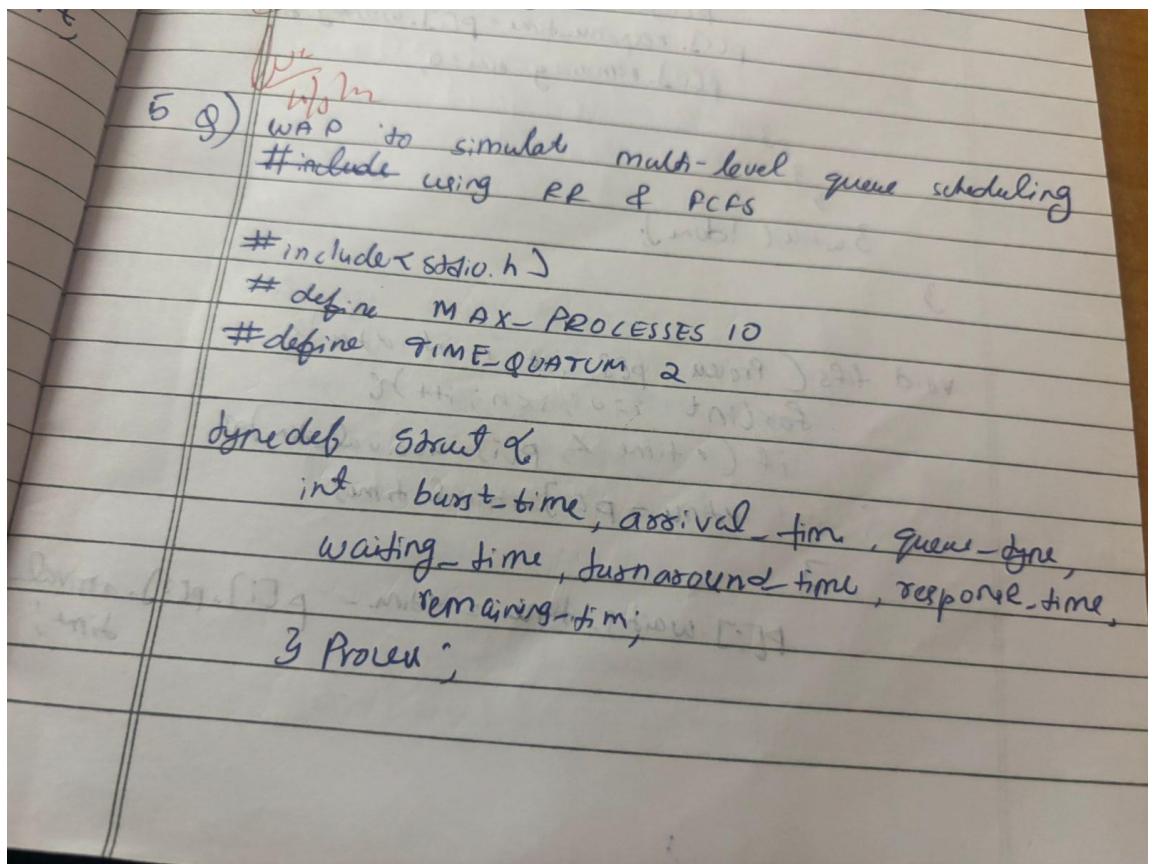
    roundRobin(n, at, bt, quant);
    return 0;
}

```

}

O/P:

```
PS C:\Users\trish\OneDrive\Desktop\OS_LAB> gcc Round.c -o Round
PS C:\Users\trish\OneDrive\Desktop\OS_LAB> ./Round
Enter number of processes: 5
Enter AT and BT for process 1: 0
8
Enter AT and BT for process 2: 5
2
Enter AT and BT for process 3: 1
7
Enter AT and BT for process 4: 6
3
Enter AT and BT for process 5: 8
5
Enter time quantum: 3
P#      AT      BT      CT      TAT      WT
1       0       8       22      22      14
2       5       2       11      6       4
3       1       7       23      22      15
4       6       3       14      8       5
5       8       5       25      17      12
Average TAT: 15.00
Average WT: 10.00
PS C:\Users\trish\OneDrive\Desktop\OS_LAB>
```



```
void roundRobin(Process p[], int n, int time_quantum, int done)
{
    int done = 0;
    for (int i = 0; i < n; i++) {
        if (p[i].remaining_time > 0) {
            done = 0;
            if (p[i].remaining_time > time_quantum) {
                p[i].remaining_time -= time_quantum;
                p[i].turnaround_time += time_quantum;
            } else {
                p[i].remaining_time = 0;
                p[i].turnaround_time += p[i].remaining_time;
            }
            p[i].waiting_time += p[i].remaining_time;
            p[i].burst_time = p[i].arrival_time;
            p[i].turnaround_time = p[i].arrival_time - p[i].burst_time;
            p[i].response_time = p[i].waiting_time;
            p[i].remaining_time = 0;
        }
    }
}

void ffs (Process p[], int n, int done) {
    for (int i = 0; i < n; i++) {
        if (i + time < p[i].arrival_time) {
            time = p[i].arrival_time;
        }
        p[i].waiting_time = time - p[i].arrival_time;
    }
}
```

PAGE NO :  
DATE :

$$p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time$$

$$p[i].response\_time = p[i].waiting\_time + p[i].burst\_time$$

+ = p[i].burst\\_time;

3

3

O/P Enter number of processes: 4  
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1

Queue of PI : 201  
P<sub>1</sub> : 102  
P<sub>2</sub> : 501  
P<sub>3</sub> : 302

Average waiting time = 4.25  
over

Queen 1 is System process  
Queen 2 is User process

Process	Waiting Time	Turn Around Time	Response Time
1	0	2	0
2	2	7	2
3	7	8	7
4	8	11	8

Average Waiting Time : 4.25

Average Turn Around Time: 4.00

Average Response Time = 4.25

Throughout = 0.36 to 0.76.

Process returned 11 (0x11) execution time = 11.52s

### Program 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 RR and FCFS scheduling for the processes in each queue.

Program:

```
#include <stdio.h>

#define MAX_PROCESSES 10
#define TIME_QUANTUM 2

typedef struct {

    int burst_time, arrival_time, queue_type, waiting_time, turnaround_time, response_time,
    remaining_time;

} Process;

void round_robin(Process processes[], int n, int time_quantum, int *time) {

    int done, i;
    do {
        done = 1;
        for (i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                done = 0;
                if (processes[i].remaining_time > time_quantum) {
                    *time += time_quantum;
                    processes[i].remaining_time -= time_quantum;
                } else {
                    *time += processes[i].remaining_time;
                    processes[i].waiting_time = *time - processes[i].arrival_time -
processes[i].burst_time;
                    processes[i].turnaround_time = *time - processes[i].arrival_time;
                }
            }
        }
    } while (!done);
}
```

```

        processes[i].response_time = processes[i].waiting_time;
        processes[i].remaining_time = 0;
    }
}

}

}

} while (!done);
}

```

```

void fcfs(Process processes[], int n, int *time) {
    for (int i = 0; i < n; i++) {
        if (*time < processes[i].arrival_time) {
            *time = processes[i].arrival_time;
        }

        processes[i].waiting_time = *time - processes[i].arrival_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        processes[i].response_time = processes[i].waiting_time;
        *time += processes[i].burst_time;
    }
}

```

```

int main() {

    Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES],
    user_queue[MAX_PROCESSES];

    int n, sys_count = 0, user_count = 0, time = 0;
    float avg_waiting = 0, avg_turnaround = 0, avg_response = 0, throughput;

    printf("Enter number of processes: ");
    scanf("%d", &n);
}

```

```

for (int i = 0; i < n; i++) {
    printf("Enter Burst Time, Arrival Time and Queue of P%d: ", i + 1);
    scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
    &processes[i].queue_type);
    processes[i].remaining_time = processes[i].burst_time;

    if (processes[i].queue_type == 1) {
        system_queue[sys_count++] = processes[i];
    } else {
        user_queue[user_count++] = processes[i];
    }
}

// Sort user processes by arrival time for FCFS
for (int i = 0; i < user_count - 1; i++) {
    for (int j = 0; j < user_count - i - 1; j++) {
        if (user_queue[j].arrival_time > user_queue[j + 1].arrival_time) {
            Process temp = user_queue[j];
            user_queue[j] = user_queue[j + 1];
            user_queue[j + 1] = temp;
        }
    }
}

printf("\nQueue 1 is System Process\nQueue 2 is User Process\n");
round_robin(system_queue, sys_count, TIME_QUANTUM, &time);
fcfs(user_queue, user_count, &time);

printf("\nProcess Waiting Time Turn Around Time Response Time\n");

```

```

for (int i = 0; i < sys_count; i++) {
    avg_waiting += system_queue[i].waiting_time;
    avg_turnaround += system_queue[i].turnaround_time;
    avg_response += system_queue[i].response_time;
    printf("%d      %d      %d\n", i + 1, system_queue[i].waiting_time,
    system_queue[i].turnaround_time, system_queue[i].response_time);
}

for (int i = 0; i < user_count; i++) {
    avg_waiting += user_queue[i].waiting_time;
    avg_turnaround += user_queue[i].turnaround_time;
    avg_response += user_queue[i].response_time;
    printf("%d      %d      %d\n", i + 1 + sys_count,
    user_queue[i].waiting_time, user_queue[i].turnaround_time, user_queue[i].response_time);
}

avg_waiting /= n;
avg_turnaround /= n;
avg_response /= n;
throughput = (float)n / time;

printf("\nAverage Waiting Time: %.2f", avg_waiting);
printf("\nAverage Turn Around Time: %.2f", avg_turnaround);
printf("\nAverage Response Time: %.2f", avg_response);
printf("\nThroughput: %.2f", throughput);
printf("\nProcess returned %d (0x%x) execution time: %.3f s\n", time, time, (float)time);

return 0;
}

```

O/P:

```
Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2

Queue 1 is System Process
Queue 2 is User Process

Process Waiting Time Turn Around Time Response Time
1      0            2                  0
2      2            7                  2
3      7            8                  7
4      8            11                 8

Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x11) execution time: 11.000 s
PS C:\Users\Admin\Desktop\1wa23cs023> 
```

```
#include<stdio.h>
#define MAX_PROCESSES 10
#define TIME_QUANTUM 2

dynedef Struct q
{
    int burst_time, arrival_time, queue_time,
        waiting_time, turnaround_time, response_time,
        remaining_time;
} Process;
```

void roundRobin ( Process p[], int n, int  
 time quantum, int \*done) {  
 int done = 0;  
 do {  
 done = 1;  
 for (i = 0; i < n; i++) {  
 if (p[i].remaining\_time > 0) {  
 done = 0;  
 if (p[i].remaining\_time > time\_quantum)  
 p[i].remaining\_time -= time\_quantum;  
 else  
 p[i].remaining\_time = 0;  
 p[i].waiting\_time += p[i].arrival\_time;  
 p[i].turnaround\_time = p[i].arrival\_time + p[i].waiting\_time;  
 p[i].response\_time = p[i].arrival\_time;  
 p[i].remaining\_time = 0;
 }
 }
 } while (!done);

PC[i]  
 PE[i]  
 \*done  
 i  
 }  
 old Enter  
 Enter  
 "D"

## Program 4

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

a)Earliest-deadline First

Program:

```
#include <stdio.h>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

struct Process {
    int id, burst_time, deadline, period;
};

void earliest_deadline_first(struct Process p[], int n, int time_limit) {
    int time = 0;
    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", p[i].id, p[i].burst_time, p[i].deadline, p[i].period);

}

printf("\nScheduling occurs for %d ms\n", time_limit);

while (time < time_limit) {

    int earliest = -1;

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

        if (p[i].burst_time > 0) {

            if (earliest == -1 || p[i].deadline < p[earliest].deadline) {

                earliest = i;
            }
        }
    }

    if (earliest == -1) break;

    printf("%dms: Task %d is running.\n", time, p[earliest].id);
    p[earliest].burst_time--;
    time++;
}

}

int main() {

    int n;

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

    struct Process processes[n];

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

```

```

for (int i = 0; i < n; i++) {
    scanf("%d", &processes[i].burst_time);
    processes[i].id = i + 1;
}

printf("Enter the deadlines:\n");
for (int i = 0; i < n; i++) {
    scanf("%d", &processes[i].deadline);
}

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

int hyperperiod = processes[0].period;
for (int i = 1; i < n; i++) {
    hyperperiod = lcm(hyperperiod, processes[i].period);
}

printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);

earliest_deadline_first(processes, n, hyperperiod);

return 0;
}

```

O/P:

```

Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3

System will execute for hyperperiod (LCM of periods): 6 ms
Earliest Deadline Scheduling:
PID    Burst    Deadline      Period
1      2        1              1
2      3        2              2
3      4        3              3

Scheduling occurs for 6 ms
0ms: Task 1 is running.
1ms: Task 1 is running.
2ms: Task 2 is running.
3ms: Task 2 is running.
4ms: Task 2 is running.
5ms: Task 3 is running.

```

Deadline

```

#include <stdio.h>
int gcd (int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
int lcm (int a, int b) {
    return (a * b) / gcd(a, b);
}

```

PAGE NO:  
DATE:

```

strand process C
    int id, burst-time, deadline, period;
    3.
    void earliest-deadline-first(Critical Process K)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            point[i] = ("PID" + Burst + Deadline + Period);
        for (int i = 0; i < n; i++)
            point[i] = i + 1;
        point[0] = 1 + id + 1 - dt[n];
        PL[i].id, PL[i].burst-time,
        PL[i].deadline, PL[i].period);
        3
        point[i] = In scheduling occurs no. of
        time-limit);
        while (sum <= time-limit) {
            int earliest = -1;
            for (int i = 0; i < n; i++)
                if ((PL[i].burst-time > 0) &&
                    (earliest == -1 || PL[i].deadline
                     < earliest))
                    earliest = i;
            3
            if (earliest == -1)
                break;
            point[i] = Task id is running
            time, PL[earliest].id];
            sum += PL[earliest].burst-time;
        }
        3
    }

```

PAGE NO:  
DATE:

Q10

Enter no. of processes : 3  
 Enter CPU burst times :  
 2 3 4  
 Enter deadlines :  
 1 2 3  
 Enter no. of time periods :  
 1 2 3

System will create no. of hypoperiods = max  
 Earliest Deadlines Scheduling

PID	Burst	Deadline	Period
1	2	1	1
2	3	2	2
3	4	3	3

Scheduling occurs in 6ms  
 0 ms : Task 1 is running  
 1 ms : Task 1 is running  
 2 ms : Task 2 is running  
 3 ms : Task 3 is running  
 4 ms : Task 2 is running  
 5 ms : Task 3 is running

## **Program 4**

b)Rate- Monotonic

program:

```
#include <stdio.h>

#define MAX_PROCESSES 10

typedef struct {

    int id;
    int burst_time;
    int period;
    int remaining_time;
    int next_deadline;
} Process;

void sort_by_period(Process processes[], int n) {

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].period > processes[j + 1].period) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
```

```

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

int calculate_lcm(Process processes[], int n) {
    int result = processes[0].period;
    for (int i = 1; i < n; i++) {
        result = lcm(result, processes[i].period);
    }
    return result;
}

double utilization_factor(Process processes[], int n) {
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += (double)processes[i].burst_time / processes[i].period;
    }
    return sum;
}

double rms_threshold(int n) {
    return n * (pow(2.0, 1.0 / n) - 1);
}

void rate_monotonic_scheduling(Process processes[], int n) {

```

```

int lcm_period = calculate_lcm(processes, n);
printf("LCM=%d\n\n", lcm_period);

printf("Rate Monotone Scheduling:\n");
printf("PID Burst Period\n");
for (int i = 0; i < n; i++) {
    printf("%d %d %d\n", processes[i].id, processes[i].burst_time, processes[i].period);
}

double utilization = utilization_factor(processes, n);
double threshold = rms_threshold(n);
printf("\n%.6f <= %.6f => %s\n", utilization, threshold, (utilization <= threshold) ? "true" :
"false");

if (utilization > threshold) {
    printf("\nSystem may not be schedulable!\n");
    return;
}

int timeline = 0, executed = 0;
while (timeline < lcm_period) {
    int selected = -1;
    for (int i = 0; i < n; i++) {
        if (timeline % processes[i].period == 0) {
            processes[i].remaining_time = processes[i].burst_time;
        }
        if (processes[i].remaining_time > 0) {
            selected = i;
            break;
        }
    }
    if (selected != -1) {
        timeline += processes[selected].period;
        executed++;
        processes[selected].remaining_time -= 1;
    }
}

```

```

    }
}

if (selected != -1) {
    printf("Time %d: Process %d is running\n", timeline, processes[selected].id);
    processes[selected].remaining_time--;
    executed++;
} else {
    printf("Time %d: CPU is idle\n", timeline);
}
timeline++;
}
}

```

```

int main() {
    int n;
    Process processes[MAX_PROCESSES];

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

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {

```

```

        scanf("%d", &processes[i].period);

    }

sort_by_period(processes, n);

rate_monotonic_scheduling(processes, n);

return 0;
}

```

o/p:

```

Enter the number of processes: 3
Enter the CPU burst times:
3 6 8
Enter the time periods:
3 4 5
LCM=60

Rate Monotone Scheduling:
PID  Burst  Period
1    3      3
2    6      4
3    8      5

4.100000 <= 0.779763 => false

System may not be schedulable!

```



double utilization = utilization + threshold (process, n)  
 double threshold = max - threshold (n);  
 point ("initial.6f" == 1.6f  $\Rightarrow$  1.5 in "utilization"  
 threshold (utilization > threshold)  $\Rightarrow$  "idle")  
 utilization > threshold &  
 point ("In System may not be  
 schedulable in")  
 return;  
 int tripulse = 0, execute = 0;  
 while (true & tripulse <= 1000000000000000000L)  
 int se = -1;  
 for (int i = 0; i < P.size(); i++) {  
 if (time[i] - PL[i].period == 0) &  
 PL[i].rem = PL[i].b + t;  
 if (PL[i].rem > 0) &  
 selected = i; // in  
 break;  
 }  
 if (selected != -1) {  
 int (selected) == 1L  
 point ("Time of execution is over");  
 timeline[PL[selected].id] = 1L;  
 PL[selected].rem = 0;  
 }  
 else {  
 point ("Time of CPU is idle", timeline);  
 timeline++;  
 }  
 tripulse++;

PAGE NO.:  
 DATE:  
 o/n  
 Entro no of process is 3  
 Entro CPU but on  
 3 6 8  
 Entro time period  
 3 4 5  
 lcm = 60  
 ems = 3  
 PID Burst Period  
 1 3 3  
 2 6 9  
 3 8 5  
 4,100 <= 0.779  $\Rightarrow$  false  
 System may not be schedulable!  
 Deadline

## **Program 5**

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

program:

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

int mutex = 1;
int full = 0;
int empty = 5;
int item = 0;

int wait(int s);
int signal(int s);
void producer();
void consumer();

int main() {
    int choice;

    printf("Producer-Consumer Problem Simulation\n");

    while (1) {
        printf("\n1. Produce\n2. Consume\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```

case 1:
    if ((mutex == 1) && (empty != 0)) {
        producer();
    } else {
        printf("Buffer is full or mutex is locked. Cannot produce.\n");
    }
    break;

case 2:
    if ((mutex == 1) && (full != 0)) {
        consumer();
    } else {
        printf("Buffer is empty or mutex is locked. Cannot consume.\n");
    }
    break;

case 3:
    exit(0);
default:
    printf("Invalid choice. Try again.\n");
}

return 0;
}

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

int signal(int s) {

```

```

return ++s;
}

void producer() {
    mutex = wait(mutex);
    empty = wait(empty);
    full = signal(full);

    item++;
    printf("Produced item %d\n", item);

    mutex = signal(mutex);
}

void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);

    printf("Consumed item %d\n", item);
    item--;

    mutex = signal(mutex);
}

```

O/P:

## Producer-Consumer Problem Simulation

1. Produce
2. Consume
3. Exit

Enter your choice: 1

Produced item 1

1. Produce
2. Consume
3. Exit

Enter your choice: 2

Consumed item 1

1. Produce
2. Consume
3. Exit

Enter your choice: 2

Buffer is empty or mutex is locked. Cannot consume.

1. Produce
2. Consume
3. Exit

Enter your choice:

```
PRODUCER CONSUMER
#include <cs.h>
#include <stdio.h>

int mutex = 1;
int full = 0;
int empty = 5;
int item = 0;
int wait(int s);
int signal(int s);
void producer();
void consumer();

int main()
{
    int choice;
    printf(" Producer - consumer problem ");
    while(1)
    {
        printf("1. Produce 2. Consume 3. Exit ");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                if(mutex == 1) || (empty == 0))
                    producer();
                else
                    printf(" Buffer is full & mutex is produced cannot produce ");
                break;
            case 2:
                if (mutex == 1) || (full == 0))
                    consumer();
                else
                    printf(" Buffer is full & mutex is consumed cannot consume ");
                break;
        }
    }
}
```

Belieb  
 point ("Buffeo" is empty or mutex is  
 locked, cannot consume item);  
 3  
 break;  
 case 3:  
 exit(0);  
 default:  
 point ("invalid choice. Try again");  
 3  
 action 0;  
 3  
 int wait(int s) do  
 action --s;  
 3  
 signal (empty);  
 & action ++s;  
 3  
 if (empty == 0) break;  
 void produce() do  
 mutex = wait (mutex);  
 empty = wait (empty);  
 full = signal (full);  
 item++;  
 produce ("Produced item " + d", item);  
 mutex = signal (mutex);  
 3  
 void consumer() do  
 mutex = wait (mutex);  
 full = wait (full);

empty = signal (empty);  
 point ("Consumed item " + d", item);  
 item--;  
 mutex = signal (mutex);  
 3  
o/p  
 , Enter choice  
 The item produced is 1000000000  
 Enter choice 2  
 consumed item  
 Enter choice  
 2  
 The port buffer is empty.  
 Enter choice  
 3  
 exiting.

## Program 6

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

Program:

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum);
void take_fork(int phnum);
void put_fork(int phnum);
void* philosopher(void* num);

int main() {
```

```

int i;
pthread_t thread_id[N];

sem_init(&mutex, 0, 1);

for (i = 0; i < N; i++)
    sem_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {
    pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
    printf("Philosopher %d is thinking\n", i + 1);
}

for (i = 0; i < N; i++)
    pthread_join(thread_id[i], NULL);

return 0;
}

void test(int phnum) {
    if (state[phnum] == HUNGRY &&
        state[LEFT] != EATING &&
        state[RIGHT] != EATING) {

        state[phnum] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
}

```

```

printf("Philosopher %d is Eating\n", phnum + 1);

sem_post(&S[phnum]);
}

}

void take_fork(int phnum) {
    sem_wait(&mutex);

    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum + 1);

    test(phnum);

    sem_post(&mutex);

    sem_wait(&S[phnum]);
    sleep(1);
}

void put_fork(int phnum) {
    sem_wait(&mutex);

    state[phnum] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);
}

```

```

test(LEFT);

test(RIGHT);

sem_post(&mutex);

}

void* philosopher(void* num) {

    int* i = (int*)num;

    while (1) {

        sleep(1);

        take_fork(*i);

        sleep(1);

        put_fork(*i);

    }

}

```

O/P:

```

PS C:\Users\Admin\Desktop\1wa23cs023> gcc dinning_phi.c -o dinning_phi
PS C:\Users\Admin\Desktop\1wa23cs023> ./dinning_phi
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes fork 1 and 2

```

	5 "	:	2 3
--	-----	---	--------

8. Dinning Philosopher

```
#include < pthread.h>
#include < semaphore.h>
#include < stdio.h>
#include <unistd.h>
#define NS
#define THINKING 2
#define HUNGRY 1
```

	3	:	0
--	---	---	---

```
#define EATING 0
#define LEFT(phnum + 0) .1. N
#define RIGHT(phnum + 1) .1. N
int state[N];
int phi[N] = {0, 1, 2, 3, 4, 5};
sem_t mutex;
sem_t SEM;

void eat(int phnum)
{
    if(state[phnum] == HUNGRY && state[LEFT(phnum)] == EATING)
        if(state[RIGHT(phnum)] != EATING)
            state[phnum] = EATING;
    sleep(2);
    printf("Philosopher %d takes lock %d and\n"
           "eats (%d) phnum %d, LEFT %d,\n"
           "phnum + 1),\n"
           "printf("Philosopher %d is Eating",\n"
           "phnum + 1);
    sem_post(&SEM[phnum]);
}

void take_fork(int phnum)
{
    sem_wait(&mutex);
    state[phnum] = HUNGRY;
    if(phi[phnum] < phi[phnum + 1])
        swap(phi[phnum], phi[phnum + 1]);
    sleep(1);
    printf("Philosopher %d is Hungry", phnum);
    sleep(1);
    sem_post(&SEM[phnum]);
}
```

PAGE NO :  
DATE :

```
void put_fork(int phnum){  
    sem_wait(&mutex);  
    state[phnum] = THINKING;  
    cout << "Philosopher " << id << " putting fork " << id  
        << " down ", phnum + 1, LEFT, phnum + 1);
```

```
philosopher(id is thinking", phnum + 1);  
det(LEFT);  
det(RIGHT);  
sem_post(&mutex);
```

}

```
void philosopher(void *num){  
    int i = (int *)num;  
    while(1){  
        sleep(1);  
        take_fork(i);  
        sleep(1);  
        put_fork(*i);  
    }  
}
```

3

O/P

Philosopher 1 is thinking  
Philosopher 2 is thinking  
Philosopher 3 is thinking

!

Philosopher 5 is thinking  
Philosopher 5 is hungry

!

4

2

Philosopher 2 is Hungry

## Program 7

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

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

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

    int alloc[n][m], max[n][m], avail[m];
    int need[n][m];

    printf("Enter allocation matrix (%d x %d):\n", n, m);
    for (i = 0; i < n; i++) {
        printf("Allocation for process %d: ", i);
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    }

    printf("Enter max matrix (%d x %d):\n", n, m);
    for (i = 0; i < n; i++) {
        printf("Max for process %d: ", i);
        for (j = 0; j < m; j++)
            scanf("%d", &max[i][j]);
    }
}
```

```
}
```

```
printf("Enter available resources (%d values): ", m);
```

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

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

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

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

```
        need[i][j] = max[i][j] - alloc[i][j];
```

```
bool finish[n];
```

```
int safeSeq[n];
```

```
int count = 0;
```

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

```
    finish[i] = false;
```

```
while (count < n) {
```

```
    bool found = false;
```

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

```
        if (!finish[i]) {
```

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

```
                if (need[i][j] > avail[j])
```

```
                    break;
```

```
                if (j == m) {
```

```
                    for (k = 0; k < m; k++)
```

```
                        avail[k] += alloc[i][k];
```

```

    safeSeq[count++] = i;
    finish[i] = true;
    found = true;
}

}

}

if (!found) {
    printf("System is not in safe state.\n");
    return 1;
}

printf("System is in safe state.\n");
printf("Safe sequence is: ");
for (i = 0; i < n; i++) {
    printf("P%d", safeSeq[i]);
    if (i != n - 1)
        printf(" -> ");
}
printf("\n");

return 0;
}

o/p:

```

```

PS C:\Users\Admin\Desktop\1wa23cs023> ./deadlock
Enter number of processes: 5
Enter number of resources: 3
Enter allocation matrix (5 x 3):
Allocation for process 0: 0 1 0
Allocation for process 1: 2 0 0
Allocation for process 2: 3 0 2
Allocation for process 3: 2 1 1
Allocation for process 4: 0 0 2
Enter max matrix (5 x 3):
Max for process 0: 7 5 3
Max for process 1: 3 2 2
Max for process 2: 9 0 2
Max for process 3: 2 2 2
Max for process 4: 4 3 3
Enter available resources (3 values): 3 3 2
System is in safe state.
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
PS C:\Users\Admin\Desktop\1wa23cs023>

```

(b) Bankers algorithm

```

int n=0, m=0, i=0, j=0, k=0;
pointf("Enter no. of processes & resources");
scanf("%d %d", &n, &m);
int alloc[n][m], max[m][n], avail[m];
int need[n][m];
printf("Enter alloc matrix:");
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        scanf("%d", &alloc[i][j]);
printf("Enter max matrix:");
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        scanf("%d", &max[i][j]);
printf("Enter available matrix:");
for(i=0; i<m; i++)
    for(j=0; j<n; j++)
        scanf("%d", &avail[i]);
for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
        if(max[i][j] - alloc[i][j] >= avail[j])
            need[i][j] = max[i][j] - alloc[i][j];
}
bool finish[n];
int safeseq[n];
int count = 0;

```

$$p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time$$

$$p[i].response\_time = p[i].waiting\_time + p[i].burst\_time$$

$$+ p[i].burst\_time$$

3

3

Average waiting / day = 4-25  
over

Queen 1 is sync process  
Queen 2 is auto process

Process	Waiting Time	Turn Around Time	Response Time
1	0	2	0
2	2	7	2
3	7	8	7
4	8	11	8

Average waiting time = 4.25

Average Turn Around Time = 4.00

Average response time = 4.25

Throughout = 0.36  
Process returned 1; (Cxxi) execution time = 11.5s

## **Program 8**

Write a C program to simulate deadlock detection

Program:

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

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

```
int main() {
```

```
    int n, m, i, j, k;
```

```
    printf("Enter number of processes and resources:\n");
```

```
    scanf("%d %d", &n, &m);
```

```
    int alloc[n][m], request[n][m], avail[m];
```

```
    bool finish[n];
```

```
    printf("Enter allocation matrix:\n");
```

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

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

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

```
    printf("Enter request matrix:\n");
```

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

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

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

```
    printf("Enter available matrix:\n");
```

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

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

```

for (i = 0; i < n; i++) {
    bool is_zero = true;
    for (j = 0; j < m; j++) {
        if (alloc[i][j] != 0) {
            is_zero = false;
            break;
        }
    }
    finish[i] = is_zero;
}

bool changed;
do {
    changed = false;
    for (i = 0; i < n; i++) {
        if (!finish[i]) {
            bool can_finish = true;
            for (j = 0; j < m; j++) {
                if (request[i][j] > avail[j]) {
                    can_finish = false;
                    break;
                }
            }
            if (can_finish) {
                for (k = 0; k < m; k++)
                    avail[k] += alloc[i][k];
                finish[i] = true;
                changed = true;
            }
        }
    }
}

```

```

        printf("Process %d can finish.\n", i);

    }

}

}

} while (changed);

bool deadlock = false;

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

    if (!finish[i]) {

        deadlock = true;

        break;

    }

}

if (deadlock)

    printf("System is in a deadlock state.\n");

else

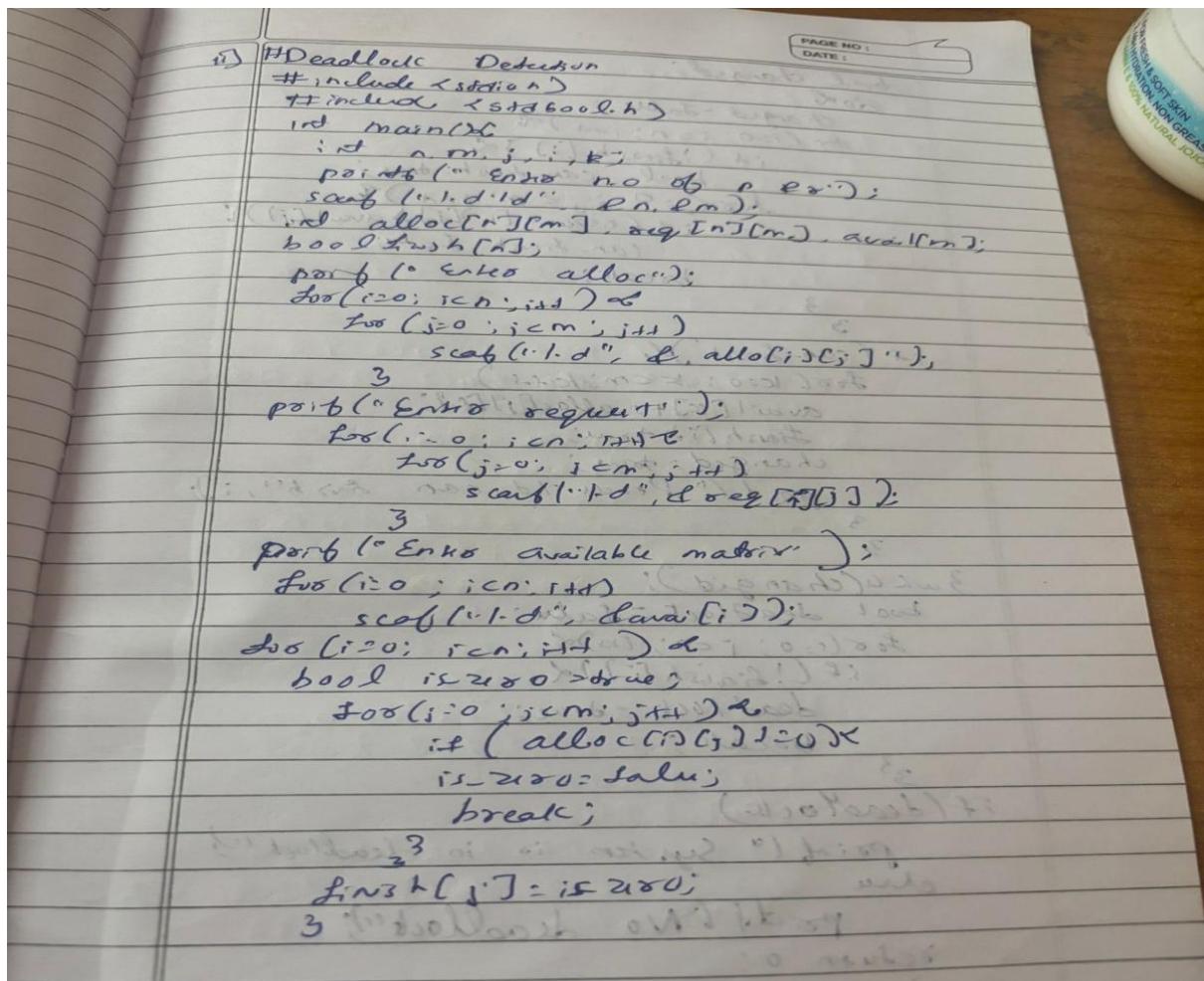
    printf("System is not in a deadlock state.\n");

return 0;
}

```

O/P:

```
PS C:\Users\Admin\Desktop\1wa23cs023> gcc deadlock2.c -o deadlock2
PS C:\Users\Admin\Desktop\1wa23cs023> ./deadlock2
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter request matrix:
0 0 0
2 0 2
0 0 1
1 0 0
0 0 2
Enter available matrix:
0 0 0
Process 0 can finish.
System is in a deadlock state.
```



```

bool changed;
do {
    charged = false;
    for (i=0; i<n; i++) {
        if (!finish[i]) {
            bool canFinish = true;
            for (j=0; j < m; j++) {
                if (sequen[i][j] > avail[j])
                    canFinish = false;
            }
            if (canFinish) {
                k = 0;
                kcm[k] = i;
                avail[k] += allocEst[i];
                finish[i] = true;
                changed = true;
            }
        }
    }
} while (!changed);
printf ("Product can finish; ");
}

3. If (changed)
    bool deadlock = false;
    for (i=0; i<n; i++)
        if (!finish[i]) deadlock = true;
    if (deadlock)
        deadlock = true;
    break;
}
if (deadlock)
    printf ("System is in deadlock");
else
    printf ("No deadlock");
return 0;

```

0/0	Enter	no	of	req	for	available
	5	3				5 units available
	Enter	allow				in stage 1
	0	1				condition 1
	2	0				stage 2
	3	0	3			stage 3
	2	1	1			stage 4
	0	0	2			stage 5
	End	sequen				available
	0	0	0			0
	2	0	2			0
	0	0	0			0
	End	available				available
	0	0	0			0
	Process	0	can	finish		
	System	is	in	deadlock	state	

## **Program 9**

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

- a) Worst-fit
- b) Best-fit
- c) First-fit

Programs:

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

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

```
struct File {  
    int file_no;  
    int file_size;  
};
```

```
void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {  
    printf("\nMemory 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 = 100000;
```

```

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);
} else {
    printf("%d\t%d\tNot Allocated\n", files[i].file_no, files[i].file_size);
}
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory 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\n",
           files[i].file_no, files[i].file_size,
           blocks[worst_fit_block].block_no,
           blocks[worst_fit_block].block_size, max_fragment);
} else {
    printf("%d\t%d\tNot Allocated\n", files[i].file_no, files[i].file_size);
}
}

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory Management Scheme - First Fit\n");
    printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");

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

```

```

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;
        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, fragment);
        allocated = 1;
        break;
    }
}

if (!allocated) {
    printf("%d\t%d\tNot Allocated\n", files[i].file_no, files[i].file_size);
}
}

void resetBlocks(struct Block blocks[], int n_blocks) {
    for (int i = 0; i < n_blocks; i++) {
        blocks[i].is_free = 1;
    }
}

int main() {
    int n_blocks, n_files, choice;

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

```

```

scanf("%d", &n_blocks);

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;
}

printf("Enter the number of files: ");
scanf("%d", &n_files);

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);
}

printf("\nChoose Memory Allocation Technique:\n");
printf("1. First Fit\n");
printf("2. Best Fit\n");
printf("3. Worst Fit\n");
printf("Enter choice (1/2/3): ");
scanf("%d", &choice);

resetBlocks(blocks, n_blocks);

```

```
switch (choice) {  
    case 1:  
        firstFit(blocks, n_blocks, files, n_files);  
        break;  
    case 2:  
        bestFit(blocks, n_blocks, files, n_files);  
        break;  
    case 3:  
        worstFit(blocks, n_blocks, files, n_files);  
        break;  
    default:  
        printf("Invalid choice\n");  
}  
  
return 0;  
}
```

O/P:

```
Enter the size of the blocks:
```

```
Block 1: 100  
Block 2: 500  
Block 3: 200  
Block 4: 300  
Block 5: 600
```

```
Enter the size of the files:
```

```
File 1: 212  
File 2: 417  
File 3: 112  
File 4: 426
```

- 1. First Fit
- 2. Best Fit
- 3. Worst Fit
- 4. Exit

```
Enter your choice: 1
```

#### Memory Management Scheme û First Fit

File_no:	File_size	Block_no:	Block_size:
1	212	2	500
2	417	5	600
3	112	3	200
4	426	-	-

- 1. First Fit
- 2. Best Fit
- 3. Worst Fit
- 4. Exit

```
Enter your choice: 2
```

#### Memory Management Scheme û Best Fit

File_no:	File_size	Block_no:	Block_size:
1	212	4	300
2	417	2	500
3	112	3	200
4	426	5	600

- 1. First Fit
- 2. Best Fit
- 3. Worst Fit
- 4. Exit

```
Enter your choice: 3
```

#### Memory Management Scheme û Worst Fit

File_no:	File_size	Block_no:	Block_size:
1	212	5	600
2	417	2	500
3	112	4	300
4	426	-	-

```

## Function to find first free block
void firstFit(Block blocks[], FILE file[], int n)
{
    for (int i = 0; i < n; i++) {
        if (blocks[i].isFree == 1)
            allocated = -1;
        for (int j = 0; j < n; j++) {
            if ((blocks[j].isFree == 0) && (blocks[j].size == file.size)) {
                fragment = blocks[i].size - blocks[j].size;
                blocks[j].isFree = 1;
                allocated = 1;
                blocks[i].size -= fragment;
                cout << "Allocated : " << i + 1 << endl;
                cout << "Block : " << j + 1 << endl;
                cout << "if (!allocated) " << endl;
                cout << "cout << (" << "Not allocated" << "); " << endl;
            }
        }
    }
}

## Worst Fit :
void worstFit(Block blocks[], FILE file[], int n)
{
    int l = 0;
    for (int i = 0; i < n; i++) {
        allocated = -1;
        worstFit = i;
        for (int j = 0; j < n; j++) {
            if ((blocks[j].isFree == 0) && (blocks[j].size >= l)) {
                allocated = 1;
                blocks[j].isFree = 1;
                blocks[i].size -= blocks[j].size;
                cout << "Allocated : " << i + 1 << endl;
                cout << "Block : " << j + 1 << endl;
                cout << "if (!allocated) " << endl;
                cout << "cout << (" << "Not allocated" << "); " << endl;
            }
        }
    }
}

```

Old

Butt1

Fileno

filesize

Blockno

Blocksize

fragments

1

212

4

300

83

2

417

2

500

23

3

112

3

200

38

4

426

5

600

129

First1

Fileno

filesize

Blockno

Blocksize

fragments

1

212

2

1500

28

2

417

5

600

123

3

112

3

200

28

4

426

Not alloc

Worst-fit

Fileno

filesize

Blockno

Blocksize

fragments

1

212

5

650

328

2

417

2

500

23

3

112

4

300

188

4

426

Not allocated

## **Program 10**

Write a C program to simulate page replacement algorithms

- a) FIFO
- b) LRU
- c) Optimal

- a) FIFO

Program:

```
#include <stdio.h>

int main() {
    int pages[100], frames[10];
    int n_pages, n_frames, i, j, k, page_faults = 0, index = 0, found;

    printf("Enter the number of pages: ");
    scanf("%d", &n_pages);

    printf("Enter the page reference string:\n");
    for (i = 0; i < n_pages; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &n_frames);

    for (i = 0; i < n_frames; i++) {
```

```

frames[i] = -1;
}

printf("\nPage\tFrames\n");

for (i = 0; i < n_pages; i++) {
    found = 0;

    for (j = 0; j < n_frames; j++) {
        if (frames[j] == pages[i]) {
            found = 1;
            break;
        }
    }

    if (!found) {
        frames[index] = pages[i];
        index = (index + 1) % n_frames;
        page_faults++;

        printf("%d\t", pages[i]);
        for (k = 0; k < n_frames; k++) {
            if (frames[k] != -1)
                printf("%d ", frames[k]);
            else
                printf("- ");
        }
        printf("\n");
    } else {

```

```

        printf("%d\tNo Page Fault\n", pages[i]);

    }

}

printf("\nTotal Page Faults: %d\n", page_faults);

return 0;
}

```

O/P:

```

PS C:\Users\Admin\Desktop\1wa23cs023> gcc FIFO.c -o FIFO
PS C:\Users\Admin\Desktop\1wa23cs023> ./FIFO
Enter the number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 4 5
Enter the number of frames: 3

Page      Frames
1          1 - -
3          1 3 -
0          1 3 0
3          No Page Fault
5          5 3 0
6          5 6 0
3          5 6 3
0          0 6 3
1          0 1 3
2          0 1 2
4          4 1 2
5          4 5 2

Total Page Faults: 11
PS C:\Users\Admin\Desktop\1wa23cs023>

```

b)LRU

Program:

```
#include <stdio.h>

int findLRU(int time[], int n) {
    int i, min = time[0], pos = 0;
    for (i = 1; i < n; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}

int main() {
    int pages[100], frames[10], time[10];
    int n_pages, n_frames, i, j, pos, page_faults = 0, counter = 0, found;

    printf("Enter the number of pages: ");
    scanf("%d", &n_pages);

    printf("Enter the page reference string:\n");
    for (i = 0; i < n_pages; i++) {
        scanf("%d", &pages[i]);
    }
```

```
}
```

```
printf("Enter the number of frames: ");  
scanf("%d", &n_frames);
```

```
for (i = 0; i < n_frames; i++) {  
    frames[i] = -1;  
}
```

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

```
for (i = 0; i < n_pages; i++) {  
    found = 0;
```

```
    for (j = 0; j < n_frames; j++) {  
        if (frames[j] == pages[i]) {  
            counter++;  
            time[j] = counter;  
            found = 1;  
            break;  
        }  
    }
```

```
    if (!found) {  
        int empty_found = 0;  
        for (j = 0; j < n_frames; j++) {  
            if (frames[j] == -1) {  
                counter++;  
                frames[j] = pages[i];
```

```

        time[j] = counter;
        page_faults++;
        empty_found = 1;
        break;
    }

}

if (!empty_found) {
    pos = findLRU(time, n_frames);
    counter++;
    frames[pos] = pages[i];
    time[pos] = counter;
    page_faults++;
}

printf("%d\t", pages[i]);
for (j = 0; j < n_frames; j++) {
    if (frames[j] != -1)
        printf("%d ", frames[j]);
    else
        printf("- ");
}
printf("\n");
} else {
    printf("%d\tNo Page Fault\n", pages[i]);
}

printf("\nTotal Page Faults: %d\n", page_faults);

```

```
return 0;  
}  
  
O/P:
```

```
PS C:\Users\Admin\Desktop\1wa23cs023> gcc LRU.c -o LRU  
PS C:\Users\Admin\Desktop\1wa23cs023> ./LRU  
Enter the number of pages: 12  
Enter the page reference string:  
1 3 0 3 5 6 3 0 1 2 4 5  
Enter the number of frames: 3  
  
Page    Frames  
1      1 - -  
3      1 3 -  
0      1 3 0  
3      No Page Fault  
5      5 3 0  
0      1 3 0  
3      No Page Fault  
5      5 3 0  
6      5 3 6  
0      1 3 0  
3      No Page Fault  
0      1 3 0  
3      No Page Fault  
0      1 3 0  
3      No Page Fault  
5      5 3 0  
0      1 3 0  
3      No Page Fault  
0      1 3 0  
0      1 3 0  
0      1 3 0  
3      No Page Fault  
0      1 3 0  
0      1 3 0  
0      1 3 0  
0      1 3 0  
0      1 3 0  
0      1 3 0  
3      No Page Fault  
5      5 3 0  
6      5 3 6  
3      No Page Fault  
0      0 3 6  
1      0 3 1  
2      0 2 1  
4      4 2 1  
5      4 2 5  
  
Total Page Faults: 10  
PS C:\Users\Admin\Desktop\1wa23cs023> []
```

c) Optimal

Program:

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

```
int predict(int pages[], int frames[], int n_pages, int index, int n_frames) {
    int i, j, farthest = index, result = -1;

    for (i = 0; i < n_frames; i++) {
        int found = 0;
        for (j = index + 1; j < n_pages; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    result = i;
                }
                found = 1;
                break;
            }
        }
        if (!found)
            return i;
    }

    return (result == -1) ? 0 : result;
}
```

```

int main() {
    int pages[100], frames[10];
    int n_pages, n_frames, i, j, page_faults = 0, filled = 0, found;

    printf("Enter the number of pages: ");
    scanf("%d", &n_pages);

    printf("Enter the page reference string:\n");
    for (i = 0; i < n_pages; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &n_frames);

    for (i = 0; i < n_frames; i++) {
        frames[i] = -1;
    }

    printf("\nPage\tFrames\n");

    for (i = 0; i < n_pages; i++) {
        found = 0;

        for (j = 0; j < n_frames; j++) {
            if (frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            page_faults++;
        }
    }
}

```

```

    }

if (!found) {
    if (filled < n_frames) {
        frames[filled++] = pages[i];
    } else {
        int pos = predict(pages, frames, n_pages, i, n_frames);
        frames[pos] = pages[i];
    }
}

page_faults++;

printf("%d\t", pages[i]);
for (j = 0; j < n_frames; j++) {
    if (frames[j] != -1)
        printf("%d ", frames[j]);
    else
        printf("- ");
}
printf("\n");
} else {
    printf("%d\tNo Page Fault\n", pages[i]);
}
}

printf("\nTotal Page Faults: %d\n", page_faults);

return 0;
}

```

O/P:

```

PS C:\Users\Admin\Desktop\1wa23cs023> gcc optimal.c -o optimal
PS C:\Users\Admin\Desktop\1wa23cs023> ./optimal
Enter the number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 4 5
Enter the number of frames: 3

Page      Frames
1         1 - -
3         1 3 -
0         1 3 0
3         No Page Fault
5         5 3 0
6         6 3 0
3         No Page Fault
0         No Page Fault
1         1 3 0
2         2 3 0
4         4 3 0
5         5 3 0

Total Page Faults: 9
PS C:\Users\Admin\Desktop\1wa23cs023>

```

13 (1) Page Replacement - FIFO

```

#include <stdio.h>
int main()
{
    int page[100], frame[100];
    int npage, nframe, i, j, k, PageFault = 0,
        index = 0, found = 0;
    printf("Enter no of pages:");
    scanf("%d", &npage);
    printf("Enter the page reference string:");
    for(i=0; i<npage; i++)
        scanf("%d", &page[i]);
    printf("Enter no of frames:");
    scanf("%d", &nframe);
    for(i=0; i<nframes; i++)
        frames[i] = -1;
    for(i=0; i<npage; i++)
    {
        found = 0;
        for(j=0; j<nframe; j++)
            if(frames[j] == page[i])
            {
                found = 1;
                break;
            }
        if(found == 0)
        {
            for(j=0; j<nframe; j++)
                if(frames[j] == -1)
                {
                    frames[j] = page[i];
                    found = 1;
                    break;
                }
            if(found == 0)
                PageFault++;
        }
    }
    printf("No of Page Faults: %d", PageFault);
}

```

13.2] Page Fault  
 #include <stdio.h>  

```

for(k=0; k < n-frames; k++) {
    if (frame[k] == -1)
        cout << "1d" << frame[k];
    else
        cout << frame[k] << " ";
}
cout << endl;
cout << "Total Page Faults : " << pagefault;
cout << endl;
    
```

in

Ques.

Enter the no of pages : 12  
 Enter the page reference string : 1 3 0 3 3 5 6 3 0 1 2 4 5  
 Enter the no of frames : 3

Page	Frames
1	1 3 0
3	3 0
0	3 0
3	No Page Fault
5	3 0
6	5 6 0
3	5 6 3
0	0 6 3
1	0 1 3
2	0 1 3
4	4 1 2
5	4 5 2

13.2] Page replacement LRU  
 #include <stdio.h>  

```

int findLRU(int time[], int n) {
    int i, minTime[0], pos = 0;
    for(i = 1; i < n; i++) {
        if (time[i] < minTime) {
            minTime[0] = time[i];
            pos = i;
        }
    }
    return pos;
}
    
```

Ans.

int main() {
 int pages[100], frames[10], tm[10];
 int nPages, nFrames, i, j, pos, pageFault = 0,
 count = 0, found;
 cout << "Enter the no of pages : ";
 cin << nPages;
 cout << "Enter the page reference string : ";
 for(i = 0; i < nPages; i++) {
 cout << pages[i];
 }
 cout << endl;
 cout << "Enter the number of frames : ";
 cin << nFrames;
 cout << endl;
 cout << "Initial frames : ";
 for(i = 0; i < nFrames; i++) {
 cout << frames[i];
 }
 cout << endl;
 cout << "Page Faults : ";
 for(i = 0; i < nPages; i++) {
 cout << endl;
 cout << "Page " << pages[i] << " is referenced at time " << tm[i];
 cout << endl;
 cout << "Frame " << frames[i] << " is replaced by page " << pages[i];
 cout << endl;
 cout << "New Frame State : ";
 for(j = 0; j < nFrames; j++) {
 cout << frames[j];
 }
 cout << endl;
 cout << endl;
 }
}

for ( $i = 0$ ;  $i < n\_pages$ ;  $i++$ )  
     found = 0;  
 for ( $j = 0$ ;  $j < n\_frames$ ;  $j++$ )  
     if (frame [ $i$ ] == page [0])  
         count++;  
     frame [ $i$ ] = page [ $i$ ];  
     time [ $i$ ] = count;  
     page\_fault += 1;  
     empty\_fault = 1;  
     break;  
  
 if (!empty\_found) {  
     pos = findLRU (time, n\_frames);  
     cout << pos  
     frame [pos] = page [ $i$ ];  
     time [pos] = count;  
     page\_fault += 0 - pos  
  
     cout << "Page " << page [ $i$ ] << endl;  
     for ( $j = 0$ ;  $j < n\_frames$ ;  $j++$ )  
         if (frame [ $j$ ] == -1)  
             point << "1." << frame [ $j$ ];  
     else  
         point << "0";  
     point << endl;  
     point << endl;

PAGE NO. :  
DATE :

points ("In Total Page Faults: 1-dn" page  
reduce 0);

2  
3  
old  
= Enter the no of pages: 12  
Enter the page references:  
1 3 0 3 5 6 3 0 1 2 4 5  
Enter the number of frames: 3  
Total Page Faults: 10

```

12 naga
    PAGE NO. _____ DATE: _____
    po::st ("Entho the number of frames");
    slab ("1.d", nPages);
    for (i=0; i<nFrames; i++)
        frame[i] = -1;
    point ("In Page & Frame (ii)");
    for (i=0; i<nPages; i++)
        found = 0;
        for (j=0; j<nFrames; j++)
            if (frame[j] == page[i])
                found = 1;
            else
                locate;
            3
            if (!found)
                if (filled < nFrames)
                    frame[filled++] = pages[i];
                else
                    pos = po::d (pages, frame, nPages,
                                i, nFrames);
                    frame[pos] = pages[i];
            3
            pageFaults++;
            point ("1.d", page[i]);
            for (j=0; j<nFrames; j++)
                if (frame[j] == -1)
                    found = 0;
                    for (k=index+1; k<nPages; k++)
                        if (frame[k] == page[i])
                            if (j > lastUsed)
                                lastUsed = j;
                            result = k;
                            found = 1;
                            locate;
                3
                if (!found)
                    return i;
            3
            return (result == -1) ? 0 : result;
        3
        main();
    int page[100], frame[10];
    int nPage, nFrame, i, j, pageFaults = 0,
        filled = 0, found;
    po::st ("Entho no of pages");
    slab ("1.d", nPages);
    for (i=0; i<nPage; i++)
        slab ("1.d", page[i]);

```

```

13 3
    Page-replacement optimal
    int predict (int pages[], int frame[], int index, int nPage)
    {
        int i, j, lastUsed, result = -1;
        for (i=0; i<nFrames; i++)
            if (frame[i] == -1)
                found = 0;
                for (j=index+1; j<nPage; j++)
                    if (frame[j] == page[i])
                        if (j > lastUsed)
                            lastUsed = j;
                            result = j;
                            found = 1;
                            locate;
            3
            if (!found)
                return i;
        3
        return (result == -1) ? 0 : result;
    3
    main();
    int page[100], frame[10];
    int nPage, nFrame, i, j, pageFaults = 0,
        filled = 0, found;
    po::st ("Entho no of pages");
    slab ("1.d", nPages);
    for (i=0; i<nPage; i++)
        slab ("1.d", page[i]);

```

$$p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time$$

$$p[i].response\_time = p[i].waiting\_time;$$

$$x \text{ time} + = p[i].burst\_time;$$

O/P Enter number of processes: 4  
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1

P2 : 1 0 2  
P3 : 5 0 1  
P4 : 3 0 2

Average Waiting Time: 4.25  
over

Queue 1 is System process  
Queue 2 is User process

Process	Waiting Time	Turn Around Time	Response Time
1	0	2	0
2	2	7	2
3	7	8	7
4	8	11	8

Average Waiting Time: 4.25

Average Turn Around Time: 7.00

Average Response Time: 4.25

Throughput: 0.36

Process returned 11 (0x11) execution time: 11.5e