Aim :- Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

Theory :-

1. Shortest Job First (Preemptive) -

Preemptive - Preemptive algorithms allow taking away CPU from process during execution. If highest priority process arrives in the system, CPU from currently executing low priority process is allocated to it.
It ensures that always highest priority process will be executing.

1. Shortest Job First (SJF) may be preemptive or non preemptive
2. Reordering the jobs so as to run the Shortest Jobs First (SJF) improves the average response time.

3. Ready queue is maintained in order of increasing job lengths.

4. When a job comes in, insert it in the ready queue based on its length.

5. SJF minimizes the average wait time because it gives service to less execution time processes before it gives service to large execution time processes.

6. While it minimizes average wait time, it may punish processes with high execution time.

7. If shorter execution time processes are in ready list, then processes with large service times tend to be left in the ready list while small processes receive service.

8. There is significant reduction in average turnaround time and average waiting time.

Example — Shortest Job First (Preemtive).

Process	BT/ET	AT	FT	WT	TT	Priority
P_1	8	0	20	12	20	3
P_2	1	1	2	0	1	1
P_3	3	2	5	0	3	2
P_4	2	3	7	2	4	3
P_5	6	4	13	3	9	4
				= 17	= 37	

Gantt Chart —

P_1	P_2	P_3	P_4	P_5	P_1

0 1 2 5 7 13 20

Average Waiting Time $= \frac{17}{5} = 3.4$.

Average Turnaround Time $= \frac{37}{5} = 7.4$.

Algorithm :-

1. Sort all the processes according to the arrival time.
2. Then select that process that has minimum arrival time and minimum burst time.
3. After completion of the process make a pool of process that arrives afterward till the completion of the previous process and select that process among the pool which is having minimum Burst time.
4. Take Process, arrival time, burst time input from user.
5. Sort the process according to arrival time and if the process has the same arrival time then sort them having less burst time.
6. Swap the process one above one in the order of execution.
7. Find the turnaround time (TT) and Waiting time (WT).
8. Find average TT and WT.

Advantages of SJF –

1. SJF better than FCFS as it reduces average waiting time.
2. Generally used for long term scheduling.
3. Propably optimal in terms of average.

Disadvantages of SJF –

1. May cause starvation.
2. Sometimes it is complicated to predict the length of the upcoming CPU request.

2. Round Robin Scheduling.

1. Round Robin Scheduling is designed especially for time-sharing systems where many processes get CPU on time sharing basis.
2. In this algorithm, a small unit of time called time quantum is defined.
3. CPU is allocated to each process for this time quantum period of time.
4. When time quantum is fixed and then processes are sheduled such that no process get CPU time more than one time quantum.
5. When time quantum expires, context switch occurs and CPU switches to other process which is scheduled next.
6. If process is executing and request for I/O the process goes in waiting (blocked) state.
7. After the completion of I/O, process gets added at the tail of ready queue.

Example — Round Robin.
(Time slice = 4ms)

Process	BT/ET	AT	FT	TT	WT
P_1	8	0	20	20	12
P_2	4	1	8	7	3
P_3	9	2	26	24	15
P_4	5	3	25	22	17.
				=	= 47

Gantt Chart —

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_3

0 4 8 12 16 20 24 25 26

Average Waiting Time $= \frac{47}{4} = 11.75$.

Average Turnaround Time $= \frac{73}{4} = 18.25$.

Algorithm :-

1. Organize all processes according to their arrival time in the ready queue. The queue structure of the ready queue is based on the FIFO structure to execute all CPU process.

2. Now, we push the first process from ready queue to execute its take for a fixed time, allocated by each process that arrives in the queue

3. If the process connot complete their task within defined time interval or slots because it is stopped by another processes that pushes from the reddy queue to execute their task due to arrival time of the next process is reached. Therefore CPU saved the previous state of the process, which helps to resume from the point where it is interrupt.

4. Similarly, the scheduler selects another process from the ready queue to execute its tasks when a process finishes its task within time slots, the process will not go for further execution because the process burst time is finished.

5. similarly, we repeat all the steps to execute the processes ~~u~~ until the work has finished.

Advantages :

1. Not face any starvation issue.
2. Easy to implement.

Disadvantages :-

1. It does not provide any special priority to execute the most important process .
2. Performance depends on time quantum .

Conclusion :- Hence , we studied to implement program for CPU scheduling algorithms such as Shortest Job First (SJF) and Round Robin with different arrival time .

need matrix = Requirement matrix — Available matrix

Human

Aim : Implement the C For Deadlock Avoidance Algorithm Bankers Algorithm.

Theory :

Deadlock Problem —

1. We know that processes needs different resources in order to complete the execution.

2. When a process requests a resource and if the resource is not available at that time, the process enters a wait state. In a multiprogramming environment it may happen with many processes.

3. There is chance that waiting processes will remain in same state and will never again change state.

4. It is because the resources they have requested are held by other waiting processes. When such a type of situation occurs then it is called as deadlock.

5. Dijkstra's Banker's algorithm is used for deadlock avoidance.

Banker's Algorithm —

i. It is applicable to the resource allocation system with multiple instances of each resource type.

ii. The algorithm requires prior knowledge of number of resources each process needs.

iii. It is less efficient than resource-allocation graph algorithm.

iv. Newly entered process should declare maximum number of instances of each resource type which it may require.

v. The request should not be more than total number of resources in the system.

vi. System checks if allocation of requested resources will leave the system is safe state. If it will the requested resources are allocated.

vii. If system determines that resources cannot be allocated as it will go in unsafe state, the requesting process should wait until other process free the resources.

Example — Max Resources matrix or Existence Matrix [13, 7, 10]

Allocation Matrix

Process \ Resources	R_1	R_2	R_3
P_1	2	1	1
P_2	7	2	3
P_3	3	2	2
P_4	1	1	3
Total Distribution	13	6	9

Max Requirement Matrix

P \ R	R_1	R_2	R_3
P_1	4	3	3
P_2	7	2	4
P_3	4	2	5
P_4	5	3	3

Need Matrix = Requirement Matrix − Allocation Matrix.

Need Matrix.

P \ R	R_1	R_2	R_3
P_1	2	2	2
P_2	0	0	1
P_3	1	0	3
P_4	4	2	0

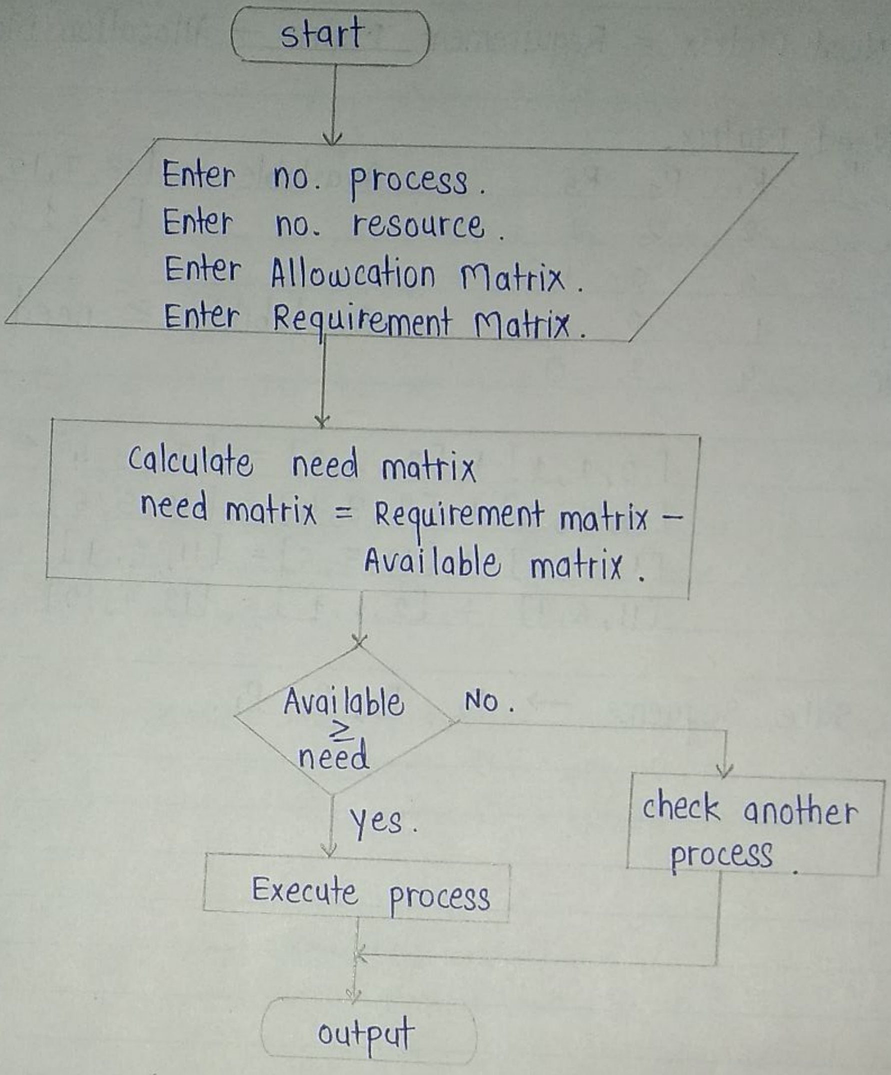
Available = [13, 7, 10] − [13, 6, 9]
= [0, 1, 1]

Availability ≥ need

[0, 1, 1] + [7, 2, 3] = [7, 3, 4] ← Newly available.
[3, 2, 2] + [7, 3, 4] = [10, 5, 6]
[1, 1, 3] + [10, 5, 6] = [11, 6, 9]
[11, 6, 9] + [2, 1, 1] = [13, 7, 10]

Safe sequence → P_2, P_3, P_4, P_1.

Flowchart :-



Conclusion :- Hence, we studied to implement the c program for deadlock avoidance using Banker's Algorithm.