**//Program for thread scheduling based on Load sharing (global queue) using FIFO on Uni-processor**

**Problem Statement:**
There are n number processes to be scheduled using FIFO. Each process comprise of two or more threads. In order to complete the processes, all the threads within the process have to completed. When a process arrives all the threads of the process arrive automatically. So there is no separate arrival time (AT) required each thread. But when a thread completes it need not be completion of the process. It is only when all the threads complete the process is completed. So there is separate FT for processes and their threads. When the last thread of a process completes, it is taken as the FT of the process.

**Scheduling**

- Get the details of the processes – no of processes, arrival times of processes and no of threads in each process.
- For each thread of a process get service time.
- Take processes one by one as per arrival time.
- Complete the threads of the chosen process by giving them the processor time equivalent to their ST.
- Note the current time as the FT of the thread.
- When the last thread of the process completes note the current time as the FT of the process.
- Display the summary of process finish times and thread finish times.

```
#include<stdio.h>
struct Thread
{
	int st; // service time
	int ft; // finish time
	int status; // thread completed - 1, not completed - 0
};

struct Process
{
	int at; // arraival time
	int nt; // no of threads
	int ntc; // no of threads completed
	int status; // process completed - 1, not-completed-0
	int ft;
	struct Thread t[5];
}plist[10];

int n,cur_time=0;

void dispatch(int*,int*);
void main()
{
	int i,j,pid,tid;
```

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

for(i=0;i<n;i++)
{
	printf("\nEnter the details of the %d process : ",i+1);
	printf("\nEnter the arrival time : ");
	scanf("%d",&plist[i].at);
	printf("\nEnter the no of threads : ");
	scanf("%d",&plist[i].nt);
	for(j=0;j<plist[i].nt;j++)
	{
		printf("\n-->Enter the service time of thread %d: ",j);
		scanf("%d",&plist[i].t[j].st);
		plist[i].t[j].status=0;
	}
	plist[i].status=0;
	plist[i].ntc=0;
}
i=0;

while(i<n) // Until all the n processess completed
{
	dispatch(&pid,&tid); // Choose the next process and the thread within that process to run
	if(pid==-1) // If no process is available at this time
	{
		cur_time++;
	}
	else
	{
		int j;
		cur_time+=plist[pid].t[tid].st;  // Update clock after the thread completed
		plist[pid].t[tid].ft=cur_time;  //Update finish time of the completed thread TID
		plist[pid].ntc++;  // Increment no of threads completed within the process PID
		plist[pid].t[tid].status=1;  // Update the status of the thread to indicate it is completed
		if(plist[pid].ntc==plist[pid].nt)  // If all the threads of the process PID are completed
		{
			 plist[pid].ft=cur_time; // Update finish time of process PID
			plist[pid].status=1; // Update completion status of process PID
			i++; // Increment number of completed processes
		}
	}
}
for(i=0;i<n;i++)
{
	printf("\nPID\tAT\tFT");
```

```
                printf("\n%d\t",i);
                printf("%d\t",plist[i].at);
                printf("%d\t",plist[i].ft);
                printf("\nThreads of process %d\n",i);
                printf("TID\tST\t\tFT");
                for(j=0;j<plist[i].nt;j++)
                {
                        printf("\n%d \t",j+1);
                        printf("%d       \t",plist[i].t[j].st);
                        printf("%d       \t",plist[i].t[j].ft);
                        printf("\n");
                }
        }
  getch();
}

void dispatch(int *pid,int *tid)
{
    *pid=*tid=-1;
        int j,k,least=cur_time,id=-1;
        for(j=0;j<n;j++)
        {
                if(plist[j].status!=1) // Process not already completed
                {
                        if(plist[j].at<=cur_time) // Process arrived within the current time
                        {
                                *pid=j;
                                for(k=0;k<plist[j].nt;k++) // Pick a thread within that process
                                {
                                        if(plist[j].t[k].status==0) // Thread not yet completed
                                        {
                                                *tid=k;
                                                return;
                                        }
                                }
                        }
                }
        }
}
/* Sample data
No. of processes 3
      AT       NO OF THREADS
P1    0        3          ST
               T1       3
               T2       2
               T3       4
```

| | | | |
|---|---|---|---|
| P2 | 3 | 2 | |
| | | T1 | 2 |
| | | T2 | 5 |
| p3 | 4 | 3 | |
| | | T1 | 1 |
| | | T2 | 3 |
| | | T3 | 2 |
| | | t4 | 2 |