Name - Ayush. D. Chougule
PRN - 1032231720
TY - CSE D1 ~~Rollno - 05~~

OS

Recall 25/8/25

Assignment - 04.

* Problem Statement : Write a program to stimulate the shortest remaining time first (SRTF) CPU scheduling algorithm in C.

* Theory :

1] Write SRTF Characteristics

→ It is a preemptive CPU scheduling algorithm where the process with the least remaining burst time is executed first if a new process arrives with a shorter remaining time, it preempts the current one. It minimizes ~~are~~ average waiting time, favour short process, but may cause starvation for longer ones. Suitable for time - Critical tasks.

eg ⇒

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

t = 0	P1 runs (remaining = 7)
t = 1	P2 arrives (BT = 4 < P1's 7)
t = 1 → 5	P2 finishes at t = 5.
t = 5	P4 (BT = 5) vs P1(7) vs P3 (9) → choose P4
t = 5 → 10	P4 finishes
t = 10 → 17	P1 finishes
t = 17 → 26	P3 finishes

Process	Completion time (CT)	TAT = CT-AT
P1	17	17
P2	5	4
P3	26	24
P4	10	7
		52

P1	P2	P4	P1	P3
0	1	5	10	17	26

Average TAT = $\frac{52}{4} = 13$

Average CT = $\frac{26}{4} = 6.5$

Eg:

Process	AT	BT	CT	TAT = CT-AT	WT = T
P1	0	6	9	9	3
P2	2	2	4	2	0
P3	4	1	5	1	0
P4	5	7	16	11	4
				23	7

t = 0 → 2 P1 runs (remaining 4)

t = 2 P2 arrives (BT = 2 < P1's 4) → preempt

t = 2 → 4 P2 finishes

t = 4 P3 arrives

t = 4 → 5 P3 finishes

t = 5 P1 (4) vs P4 (7) → Run P1

t = 5 → 9 P1 finishes

t = 9 → 16 P4 finishes.

P1	P2	P3	P1	P4
0	2	4	5	9	16

Name: Avnish. D. Chougule.

Avg TAT = 23/4 = 5.75

Avg WT = 7/4 = 1.75

* Conclusion ⇒ SRTF minimizes averages waiting time by prioritizing Shortest jobs but ~~risking~~ risks starving longer process.

* FAQ's:

1]. How is SRTF different from shortest job first.

→ SRTF is a preemptive version of shortest job first (STF). In SRTF a running process can be interrupted if a new process with a shorter remaining time arrives, while SJF is non-preemptive and runs chosen process till ~~compe~~ completion

2]. What are the major drawbacks of SRTF scheduling.

→ i) Starvation: longer process may never execute if shorter job keep arriving.

ii) Frequent context switches: Increases CPU overhead.

iii) Complex to implement: Needs accurate burst time estimation

iv) Overhead in arrival tracking: Constantly checking for the process.

3]. What is starvation in SRTF ?

→ In SRTF starvation occurs when long process wait indefinitely because shorter process keep arriving and preempt them. This continous delay prevents the longer job from getting CPU time, especially in system with frequently short-task arrivals.

CODE:

```
#include <stdio.h>

struct Process {
   int id, at, bt, rt, ct, tat, wt;
};

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

   struct Process p[n];

   for (int i = 0; i < n; i++) {
      p[i].id = i + 1;
      printf("Enter arrival time and burst time for process %d: ", p[i].id);
      scanf("%d %d", &p[i].at, &p[i].bt);
      p[i].rt = p[i].bt;
   }

   int complete = 0, time = 0, min_rt, shortest;
   int found;
   float totalWT = 0, totalTAT = 0;

   // for Gantt chart
   int ganttOrder[1000], ganttTime[1000], gcIndex = 0;
   int prev = -1;

   while (complete < n) {
      found = 0;
      min_rt = 1000000000;
      shortest = -1;

      for (int i = 0; i < n; i++) {
         if (p[i].at <= time && p[i].rt > 0 && p[i].rt < min_rt) {
            min_rt = p[i].rt;
            shortest = i;
            found = 1;
         }
```

```
    }

    if (!found) {
        time++;
        continue;
    }

    // record Gantt chart only when process changes
    if (shortest != prev) {
        ganttOrder[gcIndex] = p[shortest].id;
        ganttTime[gcIndex] = time;
        gcIndex++;
        prev = shortest;
    }

    p[shortest].rt--;
    time++;

    if (p[shortest].rt == 0) {
        complete++;
        p[shortest].ct = time;
        p[shortest].tat = p[shortest].ct - p[shortest].at;
        p[shortest].wt = p[shortest].tat - p[shortest].bt;

        totalWT += p[shortest].wt;
        totalTAT += p[shortest].tat;
    }
}

printf("\nID\tAT\tBT\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\n",
           p[i].id, p[i].at, p[i].bt,
           p[i].ct, p[i].tat, p[i].wt);
}

printf("\nAverage Turnaround Time = %.2f", totalTAT / n);
printf("\nAverage Waiting Time    = %.2f\n", totalWT / n);

// Gantt chart
printf("\nGantt Chart:\n");
for (int i = 0; i < gcIndex; i++) {
    printf("|  P%d ", ganttOrder[i]);
}
```

```
    printf("|\n0");
    for (int i = 0; i < gcIndex; i++) {
        printf("    %d", ganttTime[i]);
    }
    printf("    %d\n", time);

    return 0;
}
```

OUTPUT: