FCFS

```
#include "stdio.h"

typedef struct {
  int pid;
  int arrival_time;
  int burst_time;
  int complition_time;
  int turnaround_time;
  int waiting_time;
} Process;

void fcfs(Process processes[], int n) {
  int current_time = 0;

  for (int i=0; i<n; i++) {
    Process *p = &processes[i];

    if (p->arrival_time > current_time) {
      current_time = p->arrival_time;
    }

    p->complition_time = current_time + p->burst_time;
    p->turnaround_time = p->complition_time - p->arrival_time;
    p->waiting_time = p->turnaround_time - p->burst_time;

    current_time = p->complition_time;
  }
}

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

  Process processes[n];

  printf("Enter process details (PID, Arrival Time, Burst Time)\n");
  for (int i=0; i<n; i++) {
    Process *p = &processes[i];

    printf("Process %d: ", i);
    scanf("%d %d %d", &p->pid, &p->arrival_time, &p->burst_time);
  }

  fcfs(processes, n);

  printf("\n");
  printf("Process scheduling result (FCFS)\n");
  printf("\n");
  printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
  for (int i=0; i<n; i++) {
    Process *p = &processes[i];
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", p->pid, p->arrival_time, p->burst_time, p-
>complition_time, p->turnaround_time, p->waiting_time);
  }
  printf("\n");

  float avg_turnaround_time = 0, avg_waiting_time = 0;

  for (int i=0; i<n; i++) {
    avg_turnaround_time += processes[i].turnaround_time;
    avg_waiting_time += processes[i].waiting_time;
```

```
  }

  avg_turnaround_time /= n;
  avg_waiting_time /= n;

  printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
  printf("Average Waiting Time: %.2f\n", avg_waiting_time);

  return 0;
}
```

SJF

```
#include "stdio.h"

typedef struct {
  int pid;
  int arrival_time;
  int burst_time;
  int complition_time;
  int turnaround_time;
  int waiting_time;
} Process;

int compare_process(Process *p1, Process *p2) {}

void sjf(Process processes[], int n) {
  int current_time = 0;
  int completed_processes = 0;
  for (int i=0; i<n; i++) processes[i].complition_time = 0;

  while (completed_processes < n) {
    int sortest_job_index = -1;
    int sortest_burst_time = -1;

    for (int i=0; i<n; i++) {
      Process *p = &processes[i];
      if (p->complition_time == 0 && p->arrival_time <= current_time) {
        if (sortest_job_index == -1 || p->burst_time < sortest_burst_time) {
          sortest_job_index = i;
          sortest_burst_time = p->burst_time;
        }
      }
    }

    if (sortest_job_index == -1) {
      current_time++;
      continue;
    }

  }
}
```