**Shortest Job First (Preemptive)**

#include <stdio.h>

#include <stdbool.h>

// Process structure

typedef struct {

int pid;    // Process ID

int arrival; // Arrival time

int burst;  // Burst time

int remaining;  // Remaining burst time

bool completed; // Completion status

} Process;

// Function to find process with shortest remaining time

int findShortestJob(Process processes[], int n, int current\_time) {

int shortest = -1;

int shortest\_burst = 999999; // Assume large initial burst time

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

     if (!processes[i].completed && processes[i].arrival <= current\_time) {

         if (processes[i].burst < shortest\_burst) {

             shortest\_burst = processes[i].burst;

             shortest = i;

         }

     }

}

return shortest;

}

// Function to execute Shortest Job First (Preemptive) scheduling

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

int current\_time = 0;

int completed = 0;

while (completed != n) {

     int shortest\_index = findShortestJob(processes, n, current\_time);

     if (shortest\_index == -1) {

         printf("No process available at time %d\n", current\_time);

         current\_time++;

         continue;

     }

     // Execute the shortest job

     printf("Executing process %d at time %d\n", processes[shortest\_index].pid, current\_time);

     processes[shortest\_index].remaining--;

     current\_time++;

     // Check if process is completed

     if (processes[shortest\_index].remaining == 0) {

         processes[shortest\_index].completed = true;

         completed++;

         printf("Process %d completed at time %d\n", processes[shortest\_index].pid, current\_time);

     }

}

}

int main() {

// Example processes with different arrival times and burst times

Process processes[] = {

     {1, 0, 3, 3, false},

     {2, 2, 6, 6, false},

     {3, 4, 4, 4, false},

     {4, 6, 5, 5, false}

};

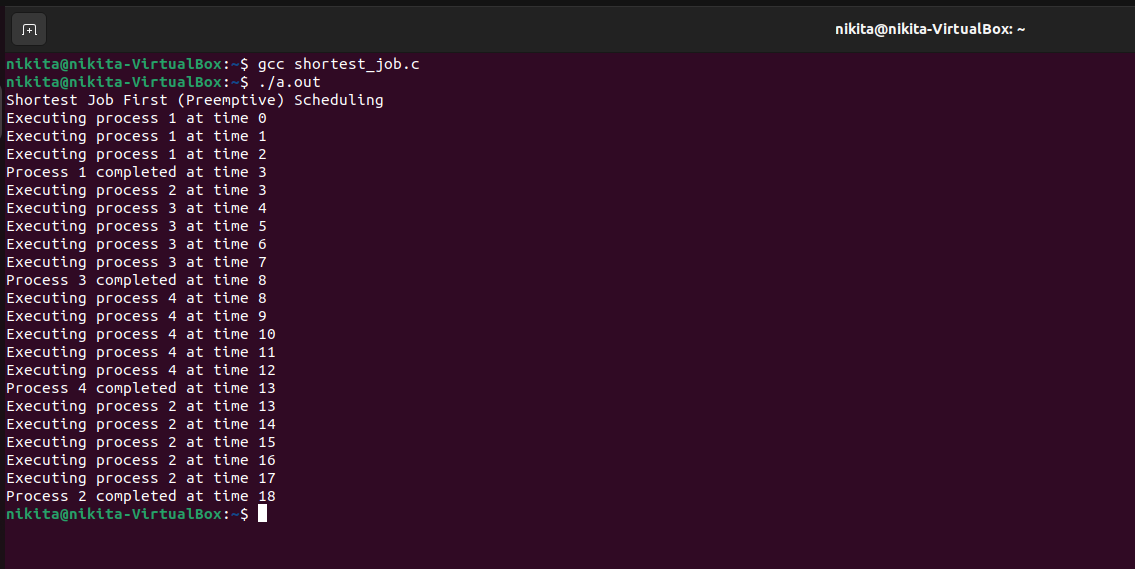
int n = sizeof(processes) / sizeof(processes[0]);

printf("Shortest Job First (Preemptive) Scheduling\n");

shortestJobFirst(processes, n);

return 0;

}



**Round Robin**

#include <stdio.h>

#include <stdbool.h>

#define QUANTUM 2  // Quantum time for Round Robin

// Process structure

typedef struct {

int pid;    // Process ID

int arrival; // Arrival time

int burst;  // Burst time

int remaining;  // Remaining burst time

bool completed; // Completion status

} Process;

// Function to execute Round Robin scheduling

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

int current\_time = 0;

int completed = 0;

while (completed != n) {

     for (int i = 0; i < n; ++i) {

         if (!processes[i].completed && processes[i].arrival <= current\_time) {

             // Execute the process for the quantum or remaining burst time

             int execute\_time = (processes[i].remaining > QUANTUM) ? QUANTUM : processes[i].remaining;

             printf("Executing process %d from time %d to %d\n", processes[i].pid, current\_time, current\_time + execute\_time);

             // Update remaining time

             processes[i].remaining -= execute\_time;

             current\_time += execute\_time;

             // Check if process is completed

             if (processes[i].remaining == 0) {

                 processes[i].completed = true;

                 completed++;

                 printf("Process %d completed at time %d\n", processes[i].pid, current\_time);

             }

         }

     }

}

}

int main() {

// Example processes with different arrival times and burst times

Process processes[] = {

     {1, 0, 4, 4, false},

     {2, 1, 3, 3, false},

     {3, 2, 1, 1, false},

     {4, 3, 2, 2, false}

};

int n = sizeof(processes) / sizeof(processes[0]);

printf("Round Robin Scheduling with Quantum = %d\n", QUANTUM);

roundRobin(processes, n);

return 0;

}

