



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GENERAL SEMESTER 2023-24

B.Tech - CSE

BCSE303P: Operating Systems Lab

Simulation of CPU scheduling algorithms

i. FCFS

```
#include <stdio.h>

typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
} process;

void fcfs(process p[], int n) {
    int waiting_time = 0;
    int turnaround_time = 0;
    printf("P\tAT\tBT\tWT\tTAT\n");

    for (int i = 0; i < n; i++) {
        waiting_time += turnaround_time - p[i].arrival_time;
        turnaround_time += p[i].burst_time;
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival_time, p[i].burst_time, waiting_time, turnaround_time);
    }

    float avg_waiting_time = (float)waiting_time / n;
    float avg_turnaround_time = (float)turnaround_time / n;
    printf("Average waiting time: %.2f\n", avg_waiting_time);
    printf("Average turnaround time: %.2f\n", avg_turnaround_time);
}

int main() {
    // Create an array of processes
```

```

process p[] = {
    {1, 0, 6},
    {2, 1, 4},
    {3, 2, 3},
    {4, 3, 7},
    {5, 4, 2},
};

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

// Call the FCFS function

fcfs(p, n);

return 0;
}

```



```

GNU nano 6.2      fcfs.c
#include <stdio.h>

typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
} process;

void fcfs(process p[], int n) {
    int waiting_time = 0;
    int turnaround_time = 0;

    printf("P\tAT\tBT\tWT\tTAT\n");

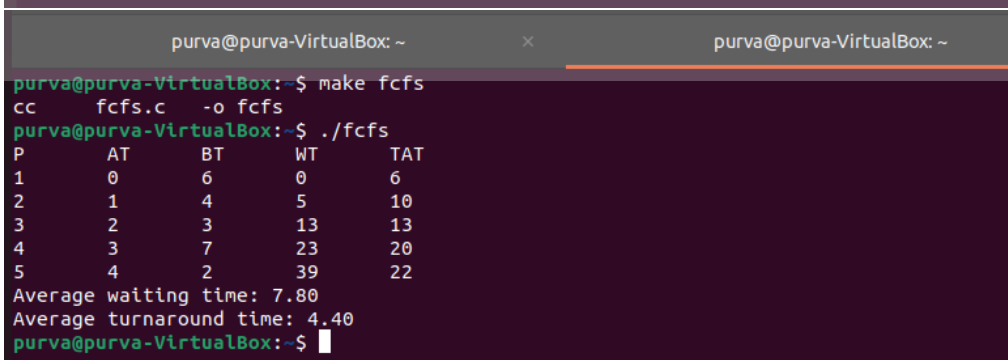
    for (int i = 0; i < n; i++) {
        waiting_time += turnaround_time - p[i].arrival_time;
        turnaround_time += p[i].burst_time;
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival_time, p[i].burst_time, waiting_time, turnaround_time);
    }

    float avg_waiting_time = (float)waiting_time / n;
    float avg_turnaround_time = (float)turnaround_time / n;
    printf("Average waiting time: %.2f\n", avg_waiting_time);
    printf("Average turnaround time: %.2f\n", avg_turnaround_time);
}

int main() {
    // Create an array of processes
    process p[] = {
        {1, 0, 6},
        {2, 1, 4},
        {3, 2, 3},
        {4, 3, 7},
        {5, 4, 2},
    };

    int n = sizeof(p) / sizeof(p[0]);
}

```



```

purva@purva-VirtualBox: ~
purva@purva-VirtualBox:~$ make fcfs
cc      fcfs.c      -o fcfs
purva@purva-VirtualBox:~$ ./fcfs
P      AT      BT      WT      TAT
1      0      6      0      6
2      1      4      5      10
3      2      3      13     13
4      3      7      23     20
5      4      2      39     22
Average waiting time: 7.80
Average turnaround time: 4.40
purva@purva-VirtualBox:~$

```

ii. SJF

```
#include <stdio.h>

typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
} process;

void sjf(process p[], int n) {
    int time = 0;
    int completed = 0;
    int min_burst_time, min_index;
    int waiting_time = 0;
    int turnaround_time = 0;
    int remaining_burst_time[n];
    int completed_time[n];

    // Initialize remaining_burst_time array
    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = p[i].burst_time;
    }
    printf("P\tAT\tBT\tWT\tTAT\n");

    while (completed != n) {
        // Find the process with the shortest remaining burst time
        min_burst_time = 1000000;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= time && remaining_burst_time[i] < min_burst_time && remaining_burst_time[i] > 0) {
                min_burst_time = remaining_burst_time[i];
                min_index = i;
            }
        }

        // Update waiting and turnaround times for the current process
        waiting_time += time - p[min_index].arrival_time;
        turnaround_time += time - p[min_index].arrival_time + p[min_index].burst_time;
```

```
// Update remaining burst time for the current process
```

```
remaining_burst_time[min_index] = 0;
```

```
// Update completion time for the current process
```

```
completed_time[min_index] = time + p[min_index].burst_time;
```

```
// Print the process ID, arrival time, burst time, waiting time, and turnaround time
```

```
printf("%d\t%d\t%d\t%d\t%d\n", p[min_index].pid, p[min_index].arrival_time, p[min_index].burst_time, waiting_time, turnaround_time);
```

```
// Update time and completed count
```

```
time += p[min_index].burst_time;
```

```
completed++;
```

```
}
```

```
float avg_waiting_time = (float)waiting_time / n;
```

```
float avg_turnaround_time = (float)turnaround_time / n;
```

```
printf("Average waiting time: %.2f\n", avg_waiting_time);
```

```
printf("Average turnaround time: %.2f\n", avg_turnaround_time);
```

```
}
```

```
int main() {
```

```
    // Create an array of processes
```

```
    process p[] = {
```

```
        {1, 0, 6},
```

```
        {2, 1, 4},
```

```
        {3, 2, 3},
```

```
        {4, 3, 7},
```

```
        {5, 4, 2},
```

```
    };
```

```
    int n = sizeof(p) / sizeof(p[0]);
```

```
    // Call the SJF function
```

```
    sjf(p, n);
```

```
    return 0;
```

```
}
```

```
GNU nano 6.2 sfj.c *
#include <stdio.h>

typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
} process;

void sjf(process p[], int n) {
    int time = 0;
    int completed = 0;
    int min_burst_time, min_index;
    int waiting_time = 0;
    int turnaround_time = 0;
    int remaining_burst_time[n];
    int completed_time[n];

    // Initialize remaining_burst_time array
    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = p[i].burst_time;
    }

    printf("P\tAT\tBT\tWT\tTAT\n");

    while (completed != n) {
        // Find the process with the shortest remaining burst time
        min_burst_time = 1000000;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= time && remaining_burst_time[i] < min_burst_time && remaini
                min_burst_time = remaining_burst_time[i];
                min_index = i;
            }

            // Update waiting and turnaround times for the current process
            waiting_time += time - p[min_index].arrival_time;
            turnaround_time += time - p[min_index].arrival_time + p[min_index].burst_time;

            remaining_burst_time[min_index] -= 1;
            if (remaining_burst_time[min_index] == 0) {
                completed++;
            }
            time++;
        }
    }

    printf("Average waiting time: %.2f\n", (float)waiting_time / n);
    printf("Average turnaround time: %.2f\n", (float)turnaround_time / n);
}
```

```
purva@purva-VirtualBox: ~
purva@purva-VirtualBox:~$ make sfj
cc sfj.c -o sfj
purva@purva-VirtualBox:~$ ./sfj
P    AT    BT    WT    TAT
1     0     6     0     6
5     4     2     2    10
3     2     3     8    19
2     1     4    18    33
4     3     7    30    52
Average waiting time: 6.00
Average turnaround time: 10.40
purva@purva-VirtualBox:~$
```

iii. Priority

```
#include <stdio.h>
```

```
typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
} process;
```

```
void priority(process p[], int n) {
    int time = 0;
    int completed = 0;
```

```
int highest_priority, highest_priority_index;
```

```
int waiting_time = 0;
```

```
int turnaround_time = 0;
```

```
int remaining_burst_time[n];
```

```
int completed_time[n];
```

```
// Initialize remaining_burst_time array
```

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

```
    remaining_burst_time[i] = p[i].burst_time;
```

```
}
```

```
printf("P\tAT\tBT\tPR\tWT\tTAT\n");
```

```
while (completed != n) {
```

```
    // Find the process with the highest priority
```

```
    highest_priority = -1;
```

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

```
        if (p[i].arrival_time <= time && p[i].priority > highest_priority && remaining_burst_time[i] > 0) {
```

```
            highest_priority = p[i].priority;
```

```
            highest_priority_index = i;
```

```
        }
```

```
    }
```

```
    // Update waiting and turnaround times for the current process
```

```
    waiting_time += time - p[highest_priority_index].arrival_time;
```

```
    turnaround_time += time - p[highest_priority_index].arrival_time + p[highest_priority_index].burst_time;
```

```
    // Update remaining burst time for the current process
```

```
    remaining_burst_time[highest_priority_index] = 0;
```

```
    // Update completion time for the current process
```

```
    completed_time[highest_priority_index] = time + p[highest_priority_index].burst_time;
```

```
    // Print the process ID, arrival time, burst time, priority, waiting time, and turnaround time
```

```
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[highest_priority_index].pid, p[highest_priority_index].arrival_time,  
p[highest_priority_index].burst_time, p[highest_priority_index].priority, waiting_time, turnaround_time);
```

```
    // Update time and completed count
```

```
    time += p[highest_priority_index].burst_time;
```

```
    completed++;
```

```
}
```

```
float avg_waiting_time = (float)waiting_time / n;
```

```

float avg_turnaround_time = (float)turnaround_time / n;

printf("Average waiting time: %.2f\n", avg_waiting_time);

printf("Average turnaround time: %.2f\n", avg_turnaround_time);

```

```

}

```

```

int main() {

```

```

    // Create an array of processes

```

```

    process p[] = {

```

```

        {1, 0, 6, 3},

```

```

        {2, 1, 4, 2},

```

```

        {3, 2, 3, 1},

```

```

        {4, 3, 7, 4},

```

```

        {5, 4, 2, 5},

```

```

    };

```

```

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

```

```

    // Call the priority function

```

```

    priority(p, n);

```

```

    return 0;

```

```

}

```



```

GNU nano 6.2 priority.c
#include <stdio.h>

typedef struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
} process;

void priority(process p[], int n) {
    int time = 0;
    int completed = 0;
    int highest_priority, highest_priority_index;
    int waiting_time = 0;
    int turnaround_time = 0;
    int remaining_burst_time[n];
    int completed_time[n];

    // Initialize remaining_burst_time array
    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = p[i].burst_time;
    }

    printf("P\tAT\tBT\tPR\tWT\tTAT\n");

    while (completed != n) {
        // Find the process with the highest priority
        highest_priority = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= time && p[i].priority > highest_priority && remaining_burs
                highest_priority = p[i].priority;
                highest_priority_index = i;
            }
        }

        // Update waiting and turnaround times for the current process
        waiting_time += time - p[highest_priority_index].arrival_time;
        turnaround_time += time - p[highest_priority_index].arrival_time + p[highest_priority_i
    }
}

```

```

purva@purva-VirtualBox:~$ make priority
cc    priority.c    -o priority
purva@purva-VirtualBox:~$ ./priority
P      AT      BT      PR      WT      TAT
1       0       6       3       0       6
5       4       2       5       2      10
4       3       7       4       7      22
2       1       4       2      21      40
3       2       3       1      38      60
Average waiting time: 7.60
Average turnaround time: 12.00
purva@purva-VirtualBox:~$

```

iv. Round Robin

```
#include <stdio.h>
```

```
typedef struct process {
```

```
    int pid;
```

```
    int arrival_time;
```

```
    int burst_time;
```

```
    int remaining_burst_time;
```

```
} process;
```

```
void round_robin(process p[], int n, int quantum) {
```

```
    int time = 0;
```

```
    int completed = 0;
```

```
    int waiting_time = 0;
```

```
    int turnaround_time = 0;
```

```
    int remaining_time[n];
```

```
    // Initialize remaining_time array
```

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

```
        remaining_time[i] = p[i].burst_time;
```

```
        p[i].remaining_burst_time = p[i].burst_time;
```

```
    }
```

```
    printf("P\tAT\tBT\tWT\tTAT\n");
```

```
    while (completed != n) {
```

```
        // Traverse all processes
```

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

```
            // Check if the process has arrived and has remaining burst time
```

```
            if (p[i].arrival_time <= time && p[i].remaining_burst_time > 0) {
```

```
                // Update remaining burst time
```

```
                if (p[i].remaining_burst_time > quantum) {
```



```

    p[i].remaining_burst_time -= quantum;

    remaining_time[i] = p[i].remaining_burst_time;

    time += quantum;
} else {

    time += p[i].remaining_burst_time;

    p[i].remaining_burst_time = 0;

    remaining_time[i] = 0;

    // Update waiting and turnaround times for the current process
    waiting_time += time - p[i].arrival_time - p[i].burst_time;

    turnaround_time += time - p[i].arrival_time;

    // Print the process ID, arrival time, burst time, waiting time, and turnaround time
    printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival_time, p[i].burst_time, waiting_time, turnaround_time);

    // Update completed count
    completed++;
}
}
}

float avg_waiting_time = (float)waiting_time / n;
float avg_turnaround_time = (float)turnaround_time / n;
printf("Average waiting time: %.2f\n", avg_waiting_time);
printf("Average turnaround time: %.2f\n", avg_turnaround_time);
}

int main() {

    // Create an array of processes
    process p[] = {
        {1, 0, 8},
        {2, 1, 4},
        {3, 2, 9},
        {4, 3, 5},
        {5, 4, 2},
    };

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

```

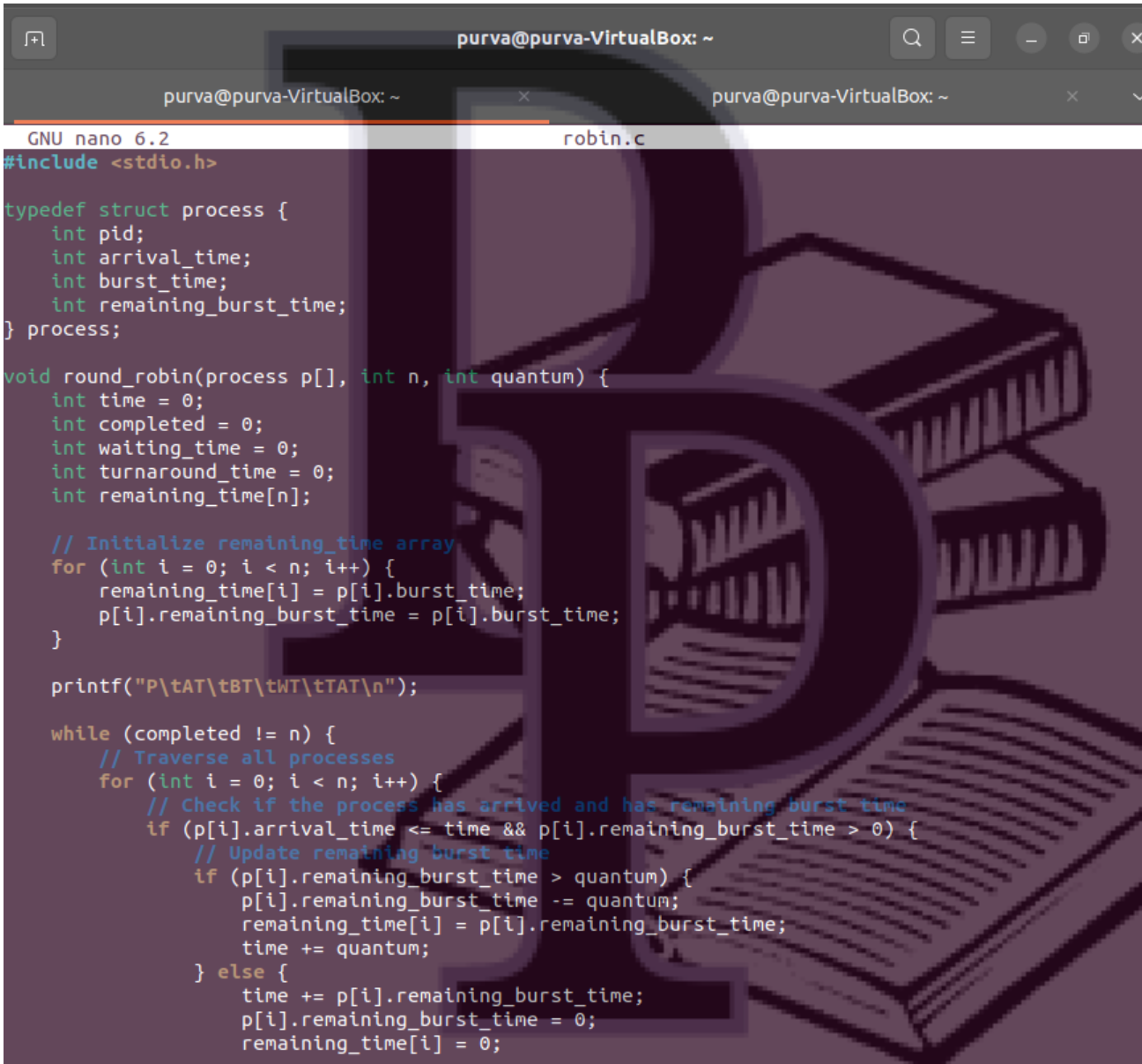
```
int quantum = 3;
```

```
// Call the round_robin function
```

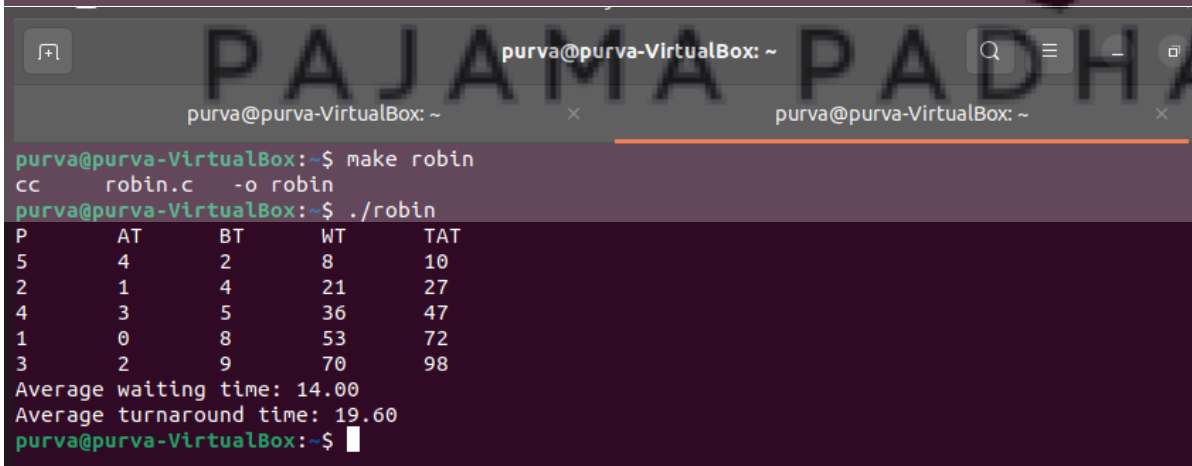
```
round_robin(p, n, quantum);
```

```
return 0;
```

```
}
```



```
purva@purva-VirtualBox: ~  
GNU nano 6.2 robin.c  
#include <stdio.h>  
  
typedef struct process {  
    int pid;  
    int arrival_time;  
    int burst_time;  
    int remaining_burst_time;  
} process;  
  
void round_robin(process p[], int n, int quantum) {  
    int time = 0;  
    int completed = 0;  
    int waiting_time = 0;  
    int turnaround_time = 0;  
    int remaining_time[n];  
  
    // Initialize remaining_time array  
    for (int i = 0; i < n; i++) {  
        remaining_time[i] = p[i].burst_time;  
        p[i].remaining_burst_time = p[i].burst_time;  
    }  
  
    printf("P\tAT\tBT\tWT\tTAT\n");  
  
    while (completed != n) {  
        // Traverse all processes  
        for (int i = 0; i < n; i++) {  
            // Check if the process has arrived and has remaining burst time  
            if (p[i].arrival_time <= time && p[i].remaining_burst_time > 0) {  
                // Update remaining burst time  
                if (p[i].remaining_burst_time > quantum) {  
                    p[i].remaining_burst_time -= quantum;  
                    remaining_time[i] = p[i].remaining_burst_time;  
                    time += quantum;  
                } else {  
                    time += p[i].remaining_burst_time;  
                    p[i].remaining_burst_time = 0;  
                    remaining_time[i] = 0;  
                    completed++;  
                    turnaround_time += time - p[i].arrival_time;  
                    waiting_time += time - p[i].burst_time;  
                }  
            }  
        }  
    }  
}
```



```
purva@purva-VirtualBox: ~  
cc robin.c -o robin  
purva@purva-VirtualBox: ~  
P   AT   BT   WT   TAT  
5   4    2    8    10  
2   1    4   21   27  
4   3    5   36   47  
1   0    8   53   72  
3   2    9   70   98  
Average waiting time: 14.00  
Average turnaround time: 19.60  
purva@purva-VirtualBox: ~
```