Operating Systems Assignment 2

Fatima Tariq (ft07200)

October 12, 2023

1 Introduction

In this assignment, the task is to develop a simulator for a process scheduler using the C programming language. The goal is to execute four different scheduling algorithms: First-Come-First-Serve (FIFO), Shortest Job First (SJF), Shortest Time-to-Completion First (STCF), and Round Robin (RR). These functions take a pointer to the queue and a pointer to the system time as parameters. Lastly, a comparison of the scheduling algorithms' performance will be carried out based on the generated metrics. The report will highlight the best-performing algorithm in terms of throughput, response time, and turnaround time.

2 Code Sections

(Comments are more elaborative in the submitted .c file)

2.1 FIFO

```
void sched_FIFO(dlq *const p_fq, int *p_time)
{
    //add code here to implement FIFO scheduling logic
    double avg_responsetime = 0.0;
    double avg_turnaround = 0.0;
    double total_responsetime = 0.0;
    double total_turnaround = 0.0;
    double avg_throughput = 0.0;
    int N = p_fq->size; //size of queue
    //printf("%d\n", N);
    dlq_node *current_node = malloc(sizeof(dlq_node)); //allocating memory
    //first node is the first one in queue since its sorted acc to arrival times
    current_node = remove_from_head(p_fq);

    dlq ready; //ready queue
    //printf("1");
    ready.head = NULL;
    ready.tail = NULL;
    int flag=0; //indicates if even one process has started executing
    int first_time = 0; //for the first process

    (*p_time)++; //since clock tick starting from 1
    while(current_node){
        int flag1=0;
        pcb *current_process = current_node->data;
        //if no processes are being executed
        if (current_process->ptimearrival!=(*p_time-1) && flag==0){
            printf("%d:idle:", (*p_time));
        } else{
            if (flag==0){
                first_time = (*p_time);
                //updating its response time (only for 1st process in scheudler)
                current_process->presponsetime = (*p_time) - current_process->
                ptimearrival;
```

```
                total_responsetime+=current_process->presponsetime;
            }
            flag=1;
            printf("%d:%s:", (*p_time), current_process->pname);

            current_process->ptimeleft--;

            if (!is_empty(&ready)){
                print_q(&ready);
                printf(":\n");
                flag1=1; }
            if (current_process->ptimeleft==0){ //if process has finished
                //updating its turnaround time
                current_process->pturnaround = (*p_time) - current_process->
                ptimearrival;
                total_turnaround+=current_process->pturnaround;
                //taking the next process from our ready queue
                current_node = remove_from_head(&ready);
                if (current_node == NULL) {
                    printf("empty:\n");
                    free(current_node); //free the allocated memory if it's NULL
                    break;
                }
                current_process = current_node->data;
                //updating its response time
                current_process->presponsetime = (*p_time+1) - current_process->
                ptimearrival;
                total_responsetime+=current_process->presponsetime;
            }
        }
        //if ready queue is empty and processes have been executed
        if (is_empty(&ready) && flag1==0){
            printf("empty:\n");}

        //temporary node
        dlq_node *n = p_fq->head;
        //if the node's time arrival == clock tick, add it in the ready queue
        while (n!=NULL && n->data->ptimearrival==(*p_time)) {
            dlq_node *node_to_move = remove_from_head(p_fq);
            add_to_tail(&ready, node_to_move);
            n = p_fq->head;
        }

        (*p_time)++; //clock tick
    }
    free(current_node);
    avg_throughput = (double)N/(((*p_time)-first_time)+1);
    avg_turnaround = total_turnaround/(double)N;
    avg_responsetime = total_responsetime/(double)N;
    printf("Average throughput: %f\n", avg_throughput);
    printf("Average response time: %f\n", avg_responsetime);
    printf("Average turnaround time: %f\n", avg_turnaround);

}
```

2.2 SJF

```
void sched_SJF(dlq *const p_fq, int *p_time)
{
    double avg_responsetime = 0.0;
```

```
double avg_turnaround = 0.0;
double total_responsetime = 0.0;
double total_turnaround = 0.0;
double avg_throughput = 0.0;
int N = p_fq->size; //size of queue

dlq_node *current_node = malloc(sizeof(dlq_node));
current_node = remove_from_head(p_fq);

dlq ready;
ready.head = NULL;
ready.tail = NULL;
int flag=0;
int first_time=0;

(*p_time)++;
while(current_node){
    int flag1=0;
    pcb *current_process = current_node->data;
    //printf("%d:", (*p_time));
    if (current_process->ptimearrival!=(*p_time-1) && flag==0){
        printf("%d:idle:", (*p_time));
    } else{
        if (flag==0){ //if it's the first process
            first_time = (*p_time);
            current_process->presponsetime = (*p_time) - current_process->
            ptimearrival;
            total_responsetime+=current_process->presponsetime;
        }
        flag=1;
        printf("%d:%s:", (*p_time), current_process->pname);
        //printf("4");
        current_process->ptimeleft--;
        //printf("%d",current_process->ptimeleft );
        if (!is_empty(&ready)){
            print_q(&ready);
            printf(":\n");
            flag1=1; }
        if (current_process->ptimeleft==0){
            current_process->pturnaround = (*p_time) - current_process->
            ptimearrival;
            total_turnaround+=current_process->pturnaround;
            current_node = remove_from_head(&ready);

            if (current_node == NULL) {
                printf("empty:\n");
                free(current_node);
                break;
            }
            current_process = current_node->data;
            current_process->presponsetime = (*p_time+1) - current_process->
            ptimearrival;
            //printf("%d\n",current_process->presponsetime);
            total_responsetime+=current_process->presponsetime;
        }
    }
    if (is_empty(&ready) && flag1==0){
        printf("empty:\n");}

    dlq_node *n = p_fq->head;
    while (n!=NULL && n->data->ptimearrival==(*p_time)) {
```

```
            dlq_node *node_to_move = remove_from_head(p_fq);
            add_to_tail(&ready, node_to_move);
            sort_by_timetocompletion(&ready);
            n = p_fq->head;
        }
        (*p_time)++;
    }
    free(current_node);
    avg_throughput = (double)N/(((*p_time)-first_time)+1);
    avg_turnaround = total_turnaround/(double)N;
        avg_responsetime = total_responsetime/(double)N;
    printf("Average throughput: %f\n", avg_throughput);
    printf("Average response time: %f\n", avg_responsetime);
        printf("Average turnaround time: %f\n", avg_turnaround);
}
```

2.3 STCF

```
void sched_STCF(dlq *const p_fq, int *p_time)
{
    double avg_responsetime = 0.0;
    double avg_turnaround = 0.0;
        double total_responsetime = 0.0;
    double total_turnaround = 0.0;
    double avg_throughput = 0.0;
    int N = p_fq->size;
    int first_time = 0;


    dlq_node *current_node = malloc(sizeof(dlq_node));
    current_node = remove_from_head(p_fq);

    dlq ready;
    //printf("1");
    ready.head = NULL;
    ready.tail = NULL;
    int flag=0;

    (*p_time)++;
    while(current_node){
        int flag1=0;
        pcb *current_process = current_node->data;
        //printf("%d:", (*p_time));
        if (current_process->ptimearrival!=(*p_time-1) && flag==0){
            printf("%d:idle:", (*p_time));
        } else{
            if (flag==0){
                first_time = (*p_time);
                printf("%d", first_time);
                current_process->presponsetime = (*p_time) - current_process->
                ptimearrival;
                total_responsetime+=current_process->presponsetime;
            }
            flag=1;
            int x = -1;
            if (ready.head != NULL && ready.head->data != NULL) {
            x = ready.head->data->ptimeleft;}
            // printf("%d\n", x);
            if (ready.head!= NULL &&current_process->ptimeleft>x){ //if current
            //process' time left is more than the next one's
```

```
            add_to_tail(&ready, current_node); //put the current process
            //back in ready queue
            current_node = remove_from_head(&ready); //start executing the
            //next process

            if (current_node == NULL) {
                printf("empty:\n");
                free(current_node);  //free memory if NULL
                break;
            }
            current_process = current_node->data;

            current_process->presponsetime = (*p_time) - current_process->
            ptimearrival;
            total_responsetime+=current_process->presponsetime;
            printf("%d\n",current_process->presponsetime);
            //sort_by_timetocompletion(&ready);

        } else if (ready.head!= NULL && current_process->ptimeleft==0){
            current_process->pturnaround = (*p_time-1) - current_process->
            ptimearrival;
            total_turnaround+=current_process->pturnaround;
            //printf("%d\n",current_process->pturnaround);
            current_node = remove_from_head(&ready);
            if (current_node == NULL) {
                printf("empty:\n");
                free(current_node);
                break;
            }
            current_process = current_node->data;
            current_process->presponsetime = (*p_time) - current_process->
            ptimearrival;
            total_responsetime+=current_process->presponsetime;

        } else if (ready.head==NULL && current_process->ptimeleft==1){
            current_process->pturnaround = (*p_time) - current_process->
            ptimearrival;
            total_turnaround+=current_process->pturnaround;
            //printf("%d\n",current_process->pturnaround);
            current_node=NULL;
            free(current_node);
        }
        printf("%d:%s:", (*p_time), current_process->pname);
        current_process->ptimeleft--;
       // printf("%d\n",current_process->ptimeleft );
        if (!is_empty(&ready)){
            sort_by_timetocompletion(&ready);
            print_q(&ready);
            printf(":\n");
            flag1=1; }
    }
    if (is_empty(&ready) && flag1==0){
        printf("empty:\n");}

    dlq_node *n = p_fq->head;
    while (n!=NULL && n->data->ptimearrival==(*p_time)) {
        dlq_node *node_to_move = remove_from_head(p_fq);
        add_to_tail(&ready, node_to_move);
        sort_by_timetocompletion(&ready);
        n = p_fq->head;
    }
```

```
        (*p_time)++;
    }
    free(current_node);
    avg_throughput = (double)N/(((*p_time)-first_time));
    avg_turnaround = total_turnaround/(double)N;
        avg_responsetime = total_responsetime/(double)N;
    printf("Average throughput: %f\n", avg_throughput);
    printf("Average response time: %f\n", avg_responsetime);
        printf("Average turnaround time: %f\n", avg_turnaround);
}
```

2.4 RR

```
void sched_RR(dlq *const p_fq, int *p_time)
{
    double avg_responsetime = 0.0;
    double avg_turnaround = 0.0;
    double total_responsetime = 0.0;
    double total_turnaround = 0.0;
    double avg_throughput = 0.0;
    int N = p_fq->size;
    int first_time = 0;

    dlq_node *current_node = malloc(sizeof(dlq_node));
    current_node = remove_from_head(p_fq);

    dlq ready;
    //printf("1");
    ready.head = NULL;
    ready.tail = NULL;
    int flag=0;

    (*p_time)++;
    while(current_node){
        int flag1=0;
        pcb *current_process = current_node->data;
        if (current_process->ptimearrival!=(*p_time-1) && flag==0){
            printf("%d:idle:", (*p_time));
        } else{
            if (flag==0){
                first_time = (*p_time);
                current_process->presponsetime = (*p_time) - current_process->
                ptimearrival;
                total_responsetime+=current_process->presponsetime;
                printf("%d\n",current_process->presponsetime);
            }
            flag=1;
            printf("%d:%s:", (*p_time), current_process->pname);

            if (!is_empty(&ready)){
                print_q(&ready);
                printf(":\n");
                flag1=1; }

            if (current_process->ptimeleft>1){ //1 is the time slice, if its
            //been executed for 1 clock tick and time remaining is more than
            //time slice then move to next process
                current_process->ptimeleft--;
```

```
                //current_node = remove_from_head(&ready);
                add_to_tail(&ready, current_node); //add current back to ready
                current_node = remove_from_head(&ready);
                if (current_node == NULL) {
                    printf("empty:\n");
                    free(current_node); //free the allocated memory if it's NULL
                    break;
                }
                current_process = current_node->data;
                if (current_process->presponsetime==0){
                    current_process->presponsetime = (*p_time+1)-current_process
                    ->ptimearrival;
                    printf("%d\n",current_process->presponsetime);
                    total_responsetime+=current_process->presponsetime;

                }

            } else{
                current_process->pturnaround = (*p_time) - current_process->
                ptimearrival;
                total_turnaround+=current_process->pturnaround;
                current_node = remove_from_head(&ready);
                if (current_node == NULL) {
                    printf("empty:\n");
                    free(current_node); //free the allocated memory if it's NULL
                    break;
                }
                current_process = current_node->data;
            }

        }
        if (is_empty(&ready) && flag1==0){
            printf("empty:\n");}

        dlq_node *n = p_fq->head;
        while (n!=NULL && n->data->ptimearrival==(*p_time)) {
            dlq_node *node_to_move = remove_from_head(p_fq); //temp node
            add_to_tail(&ready, node_to_move);
            n = p_fq->head;
        }

        (*p_time)++;
    }
    free(current_node);
    avg_throughput = (double)N/(((*p_time)-first_time)+1);
    avg_turnaround = total_turnaround/(double)N;
    avg_responsetime = total_responsetime/(double)N;
    printf("Average throughput: %f\n", avg_throughput);
    printf("Average response time: %f\n", avg_responsetime);
    printf("Average turnaround time: %f\n", avg_turnaround);
}
```

2.5 Changes in given code

I have modified the queue structure (dlq) to add size. This is being updated in adding to tail and removing from head functions. I have used this size for time calculations in my functions (as N).
In the pcb structure, I have added turnaround time and response time and then updated it in my respective scheduler functions for each process.

3 Makefile

```
hw2: hw2.c

build:
    gcc -Wall -o hw2 hw2.c


rebuild:
    clean
    gcc -Wall -o hw2 hw2.c

clean:
    rm -f hw2.c


run:
    ./hw2
```

4 Performance Metrics

4.1 Test Case 1

```
3
P1:12:7:3
P2:15:3:5
P3:1:6:2
```

Table 1: Performance Metrics for Scheduling Algorithms (Test Case 1)

Scheduling Algorithm	Average Throughput	Average Response Time	Average Turnaround Time
FIFO	0.187500	6.000000	10.333333
SJF	0.187500	4.666667	9.000000
STCF	0.187500	4.666667	9.000000
RR	0.187500	2.000000	12.333333

4.2 Test Case 2

```
6
P1:1:5:0
P2:2:7:2
P3:3:6:3
P4:4:9:4
P5:5:8:5
P6:6:4:7
```

Table 2: Performance Metrics for Scheduling Algorithms (Test Case 2)

Scheduling Algorithm	Average Throughput	Average Response Time	Average Turnaround Time
FIFO	0.153846	13.666667	19.166667
SJF	0.153846	11.333333	16.833333
STCF	0.153846	11.333333	16.833333
RR	0.153846	4.333333	26.000000

4.3 Test Case 3

```
6
```

```
P1:1:6:0
P2:2:12:2
P3:3:8:4
P4:4:15:5
P5:5:5:7
P6:6:10:9
```

Table 3: Performance Metrics for Scheduling Algorithms (Test Case 3)

Scheduling Algorithm	Average Throughput	Average Response Time	Average Turnaround Time
FIFO	0.107143	19.333333	27.666667
SJF	0.107143	14.666667	23.000000
STCF	0.107143	15.000000	22.666667
RR	0.107143	3.500000	36.333333

4.4 Test Case 4

```
8
P1:1:10:0
P2:2:15:2
P3:3:8:4
P4:4:12:6
P5:5:6:9
P6:6:4:11
P7:7:7:13
P8:8:9:15
```

Table 4: Performance Metrics for Scheduling Algorithms (Test Case 4)

Scheduling Algorithm	Average Throughput	Average Response Time	Average Turnaround Time
FIFO	0.112676	28.625000	36.500000
SJF	0.112676	19.500000	27.375000
STCF	0.112676	19.750000	27.250000
RR	0.112676	4.500000	49.625000

5 Analysis of Performance Metrics

5.1 Throughput

The throughput is the same for all the schedulers. As the number of processes increased, the throughput also increased.

5.2 Response Time

Overall, Round Robin had the best response time for all the test cases while First In First Out (FIFO) had the worst one. Shortest Job First (SJF) and Shortest Time To Completion (STCF) had almost the same response time.

5.3 Turnaround Time

Shortest Time To Completion (STCF) had the best turnaround time (evident in the last two test cases) although Shortest Job First's (SJF) was also close. In terms of worst, it was Round Robin.