Operating Systems Homework-2

Sarim Tahir (st07112)

October 12, 2023

1 First-In First-Out (FIFO) Algorithm

```c
void sched_FIFO(dlq *const p_fq, int *p_time)
{
    dlq nq;
    nq.tail = NULL;
    nq.head = NULL;
    dlq_node* node = p_fq->head;
    dlq_node* node2 = node->pbck;
    while (node){
        if (*p_time < node->data->ptimearrival){
            printf("%d:idle:empty:\n",((*p_time)++)+1);
        }
        else{
            if (nq.head){
                remove_from_head(&nq);
            }
            node->data->ptimestart = *p_time;
            while (node->data->ptimeleft > 0) {
                while (node2){
                    if ((node2->data->ptimearrival)== *p_time){
                        add_to_tail(&nq, get_new_node(node2->data))
    ;
                        node2 = node2->pbck;
                    }
                    else {
                        break;
                    }
                }
                printf("%d:%s:", ((*p_time)++)+1, node->data->pname
    );
                if (nq.head){
                    print_q(&nq);
                }
                else{
                    printf("empty");
                }
                printf(":\n");
                node->data->ptimeleft = node->data->ptimeleft - 1;
            }
            node->data->ptimend = *p_time;
            node = nq.head;
        }
    }
}
```

Listing 1: FIFO Algorithm

The code above is the implementation of the FIFO algorithm. A loop runs for the current node/process till their are none. while running it add new nodes to the ready queue(nq) if the system time is equal to their arrival time. when a nodes time left is 0 it is replaced by the head of nq.

1.1 Analysis of FIFO Algorithm

The table below shows the average throughput, average turnaround time and average response time for the FIFO algorithm for test case 0, 2, 5, 10, and 13.

Test Case	Average Throughput (ms)	Average Turnaround Time (ms)	Average Response Time (ms)
0/2	0.167	10.333	5.000
5	0.154	19.167	12.667
10	0.107	27.667	18.333
13	0.113	36.500	27.625

2 Shortest Job First (SJF) Algorithm

Below is an example of C code with syntax highlighting:

```c
void sched_SJF(dlq *const p_fq, int *p_time)
{
    dlq nq;
    nq.tail = NULL;
    nq.head = NULL;
    dlq_node* node = p_fq->head;
    dlq_node* node2 = node->pbck;
    add_to_tail(&nq, get_new_node(node->data));
    while (node){
        if (*p_time < node->data->ptimearrival){
            printf("%d:idle:empty:\n",((*p_time)++)+1);
        }
        else{
            if (nq.head){
                remove_from_head(&nq);
            }
            node->data->ptimestart = *p_time;
            while (node->data->ptimeleft > 0) {
                while (node2){
                    if ((node2->data->ptimearrival)== *p_time){
                        add_to_tail(&nq, get_new_node(node2->data))
    ;
                        sort_by_timetocompletion(&nq);
                        node2 = node2->pbck;
                    }
                    else {
                        break;
                    }
                }
                printf("%d:%s:", ((*p_time)++)+1, node->data->pname
    );
                if (nq.head){
                    print_q(&nq);
                }
                else{
                    printf("empty");
                }
                printf(":\n");
                node->data->ptimeleft = node->data->ptimeleft - 1;
            }
            node->data->ptimend = *p_time;
            node = nq.head;
        }
    }
}
```

Listing 2: SJF Algorithm

The code above is the implementation of the SJF algorithm. It is implemented similar to FIFO it just sorts the queue by time to completion when a new node is added.

2.1 Analysis of SJF Algorithm

The table below shows the average throughput, average turnaround time and average response time for the SJF algorithm for test case 0, 2, 5, 10, and 13.

Test Case	Average Throughput (ms)	Average Turnaround Time (ms)	Average Response Time (ms)
0/2	0.167	9.000	3.667
5	0.154	16.833	10.333
10	0.107	23.000	13.667
13	0.113	27.375	18.500

3 Shortest Time Completion First (STCF) Algorithm

```
void sched_STCF(dlq *const p_fq, int *p_time)
{
    dlq nq;
    nq.tail = NULL;
    nq.head = NULL;
    dlq_node* node = p_fq->head;
    dlq_node* node2 = node->pbck;
    add_to_tail(&nq, get_new_node(node->data));
    while (node){
        if (*p_time < node->data->ptimearrival){
            printf("%d:idle:empty:\n",((*p_time)++)+1);
        }
```

```c
        else{
            if (nq.head){
                remove_from_head(&nq);
            }
            if (node->data->ptimestart == -1){
                node->data->ptimestart = *p_time;
            }
            while (node->data->ptimeleft > 0) {
                while (node2){
                    if ((node2->data->ptimearrival)== *p_time){
                        add_to_tail(&nq, get_new_node(node2->data))
    ;
                        sort_by_timetocompletion(&nq);
                        node2 = node2->pbck;
                    }
                    else {
                        break;
                    }
                }
                if ((nq.head)&&(node->data->ptimeleft > nq.head->
    data->ptimeleft)){
                    add_to_tail(&nq, get_new_node(node->data));
                    sort_by_timetocompletion(&nq);
                    break;
                }
                printf("%d:%s:", ((*p_time)++)+1, node->data->pname
    );
                if (nq.head){
                    print_q(&nq);
                }
                else{
                    printf("empty");
                }
                printf(":\n");
                node->data->ptimeleft = node->data->ptimeleft - 1;
            }
            if (node->data->ptimeleft == 0){
                node->data->ptimend = *p_time;
            }
            node = nq.head;
        }
    }
}
```

Listing 3: STCF Algorithm

The code above is the implementation of the STCF algorithm. IT is implemented similar to SJF but it compares the ready queue head time left with current process' time left and does context witching if necessary.

3.1 Analysis of STCF Algorithm

The table below shows the average throughput, average turnaround time and average response time for the STCF algorithm for test case 0, 2, 5, 10, and 13.

Test Case	Average Throughput (ms)	Average Turnaround Time (ms)	Average Response Time (ms)
0/2	0.167	9.000	3.667
5	0.154	16.833	10.333
10	0.107	22.667	12.500
13	0.113	27.250	17.875

4 Round Robin (RR) Algorithm

```c
void sched_RR(dlq *const p_fq, int *p_time)
{
    dlq nq;
    nq.tail = NULL;
    nq.head = NULL;
    dlq_node* node = p_fq->head;
    dlq_node* node2 = node->pbck;
    add_to_tail(&nq, get_new_node(node->data));
    while (node){
        if (*p_time < node->data->ptimearrival){
            printf("%d:idle:empty:\n",((*p_time)++)+1);
        }
        else{
            if (nq.head){
                remove_from_head(&nq);
            }
            if (node->data->ptimestart == -1){
                node->data->ptimestart = *p_time;
            }
            while (node->data->ptimeleft > 0) {
                while (node2){
                    if ((node2->data->ptimearrival)== *p_time){
                        add_to_tail(&nq, get_new_node(node2->data))
    ;
                        node2 = node2->pbck;
                    }
                    else {
                        break;
                    }
                }
                printf("%d:%s:", ((*p_time)++)+1, node->data->pname
    );
                if (nq.head){
                    print_q(&nq);
                }
                else{
                    printf("empty");
                }
                printf(":\n");
                node->data->ptimeleft = node->data->ptimeleft - 1;
                if ((nq.head)&&(node->data->ptimeleft > 0)){
                    add_to_tail(&nq, get_new_node(node->data));
                    break;
                }
            }
            if (node->data->ptimeleft == 0){
                node->data->ptimend = *p_time;
            }
            node = nq.head;
        }
    }
}
```

Listing 4: RR Algorithm

The code above is the implementation of the RR algorithm. Similar to STCF but it doesn't sort the ready queue by time to completion and it does context switching in every cycle regardless of time left as long as there is a process in ready queue.

4.1 Analysis of RR Algorithm

The table below shows the average throughput, average turnaround time and average response time for the RR algorithm for test case 0, 2, 5, 10, and 13.

Test Case	Average Throughput (ms)	Average Turnaround Time (ms)	Average Response Time (ms)
0/2	0.167	12.333	1.000
5	0.154	26.000	2.500
10	0.107	36.333	2.500
13	0.113	49.625	3.50

5 Comparison of Algorithms

The table below shows the overall average throughput, average turnaround time and average response time for all algorithms for test case 0, 2, 5, 10, and 13.

Algorithm	Average Throughput (ms)	Average Turnaround Time (ms)	Average Response Time (ms)
FIFO	0.142	20.800	13.725
SJF	0.142	17.042	9.967
STCF	0.142	16.950	9.61
RR	0.142	27.325	2.100

From the above table we have the following insights:

- All algorithms exhibit the same average throughput. This uniformity arises from the calculation of average throughput, which involves dividing the total number of completed processes by the aggregate time taken. As all algorithms accomplish an identical number of processes within the same timeframe, the average throughput remains constant.

- The STCF algorithm boasts the lowest average turnaround time. This achievement can be attributed to the STCF algorithm's strategy of prioritizing processes with the shortest remaining execution time. By consistently scheduling processes with the least time left to completion (via context switching), it optimizes the average turnaround time.
- The RR algorithm achieves the lowest average response time. This outcome can be traced back to the RR algorithm's approach of scheduling processes in a round-robin fashion. Each process is allotted a time quantum of 1ms, after which the next process in the queue is scheduled. This equitable distribution of scheduling ensures that processes receive fair treatment, leading to the lowest average response time.

6 Appendix

6.1 MakeFile

```
CC = gcc
CFLAGS = -Wall

SRCS = os2.c

TARGETS = $(SRCS:.c=)

.PHONY: all clean

all: $(TARGETS)

os2: os2.c
	$(CC) $(CFLAGS) -o os2 os2.c

clean:
	rm -f $(TARGETS)
```

Listing 5: MakeFile