CS232 Operating Systems

Assignment 02: Simulate a scheduler

First-Come-First-Serve (FIFO) Scheduling:

Code:

```
//implement the FIFO scheduling code
void sched_FIFO(dlq *const p_fq, int *p_time)
{
    dlq q;
    q.head = NULL;
    q.tail = NULL;

    dlq_node *proc;
    proc = remove_from_head(p_fq);

    while (1) {
        ++*p_time;
        if (is_empty(p_fq)) {
            if (is_empty(&q) && !(proc->data->ptimeleft)) {
                break;
            }
        } else {
            if (p_fq->head->data->ptimearrival < *p_time) {
                add_to_tail(&q, remove_from_head(p_fq));
            }
        }
        printf("%d:", *p_time);

        if (proc->data->ptimearrival < *p_time) {
            --proc->data->ptimeleft;
            printf("%s:", proc->data->pname);
        } else {
            printf("idle:");
        }
        if (is_empty(&q)) {
            printf("empty:\n");
        } else {
            print_q(&q);
            printf(":\n");
            if (proc->data->ptimeleft == 0) {
```

```
                proc = remove_from_head(&q);
            }
        }
    }
    free(proc);
}
```

Explanation:

- It manages two queues: p_fq (processes awaiting execution) and q (processes currently in execution).
- The code operates in a loop, incrementing a time counter with each iteration.
- It checks conditions for exiting the loop: if p_fq is empty, q is empty, and the current process has no time left.
- If p_fq is not empty, it checks if the next process is ready for scheduling based on its arrival time.
- If a process is ready, it moves from p_fq to q.
- It prints the current time and the state (executing or idle) of the processes, and updates the time remaining for the executing process.
- If q is empty, it prints "empty"; otherwise, it lists the processes in q.
- If the executing process finishes, it is removed from q.
- The loop continues until scheduling is complete, and memory for proc is freed.

Shortest Job First (SJF) Scheduling:

Code:

```
//implement the SJF scheduling code
void sched_SJF(dlq *const p_fq, int *p_time)
{
    dlq q;
    q.head = NULL;
    q.tail = NULL;

    dlq_node *proc;
    proc = remove_from_head(p_fq);

    while (1) {
        ++*p_time;

        if (is_empty(p_fq)) {
            if (is_empty(&q) && !(proc->data->ptimeleft)) {
                break;
            }
```

```
        } else {
            if (p_fq->head->data->ptimearrival < *p_time) {
                add_to_tail(&q, remove_from_head(p_fq));
                sort_by_timetocompletion(&q);
            }
        }

        printf("%d:", *p_time);

        if (proc->data->ptimearrival < *p_time) {
            --proc->data->ptimeleft;
            printf("%s:", proc->data->pname);
        } else {
            printf("idle:");
        }

        if (is_empty(&q)) {
            printf("empty:\n");
        } else {
            print_q(&q);
            printf(":\n");
            if (proc->data->ptimeleft == 0) {
                proc = remove_from_head(&q);
            }
        }
    }
    free(proc);
}
```

Explanation:

- It maintains two queues: `p_fq` for processes awaiting execution and `q` for processes currently in execution.
- The code iterates in a loop, incrementing a time counter with each iteration.
- It checks conditions for exiting the loop: if `p_fq` is empty, `q` is empty, and the current process has no time left.
- If `p_fq` isn't empty, it checks if the next process is ready for scheduling based on its arrival time.
- If a process is ready, it moves from `p_fq` to `q` and sorts the queue based on time to completion.
- The code prints the current time and the state (executing or idle) of the processes, and updates the time remaining for the executing process.
- If `q` is empty, it prints "empty"; otherwise, it lists the processes in `q`.
- If the executing process finishes, it is removed from `q`.

- The loop continues until scheduling is complete, and memory for `proc` is freed.

Shortest Tiem-to-Completion First (STCF) Scheduling:

Code:

```
//implement the STCF scheduling code
void sched_STCF(dlq *const p_fq, int *p_time)
{
    dlq q;
    q.head = NULL;
    q.tail = NULL;

    dlq_node *proc;
    proc = remove_from_head(p_fq);

    while (1) {
        ++*p_time;

        if (is_empty(p_fq)) {
            if (is_empty(&q) && !(proc->data->ptimeleft)) {
                break;
            }
        } else {
            if (p_fq->head->data->ptimearrival < *p_time) {
                add_to_tail(&q, remove_from_head(p_fq));
            }
        }

        printf("%d:", *p_time);
        sort_by_timetocompletion(&q);

        if (proc->data->ptimearrival < *p_time) {
            if (!is_empty(&q) && (q.head->data->ptimeleft < proc-
>data->ptimeleft)) {
                add_to_tail(&q, proc);
                proc = remove_from_head(&q);
            }
            --proc->data->ptimeleft;
            printf("%s:", proc->data->pname);
        } else {
            printf("idle:");
        }
```

```
        if (is_empty(&q)) {
            printf("empty:\n");
        } else {
            sort_by_timetocompletion(&q);
            print_q(&q);
            printf(":\n");
            if (proc->data->ptimeleft == 0) {
                proc = remove_from_head(&q);
            }
        }
    }

    free(proc);
}
```

Explanation:

- It maintains two queues: p_fq for processes awaiting execution and q for processes currently in execution.
- The code iterates in a loop, incrementing a time counter with each iteration.
- It checks conditions for exiting the loop: if p_fq is empty, q is empty, and the current process has no time left.
- If p_fq isn't empty, it checks if the next process is ready for scheduling based on its arrival time.
- If a process is ready, it moves from p_fq to q.
- The queue q is sorted based on time to completion.
- The code prints the current time and the state (executing or idle) of the processes, and updates the time remaining for the executing process.
- If a shorter job arrives while a process is already executing, it is scheduled in place of the current process.
- If the executing process finishes, it is removed from q.
- The loop continues until scheduling is complete, and memory for proc is freed.

Round Robin (RR) Scheduling:

Code:

```
//implement the RR scheduling code
void sched_RR(dlq *const p_fq, int *p_time)
{
    dlq q;
    q.head = NULL;
    q.tail = NULL;
```

```
dlq_node *proc;
proc = remove_from_head(p_fq);

int time_slice = 1;
int proc_time = 0;

while (1) {
    ++*p_time;

    if (is_empty(p_fq)) {
        if (!(proc->data->ptimeleft) && is_empty(&q)) {
            break;
        }
    } else {
        if (p_fq->head->data->ptimearrival < *p_time) {
            add_to_tail(&q, remove_from_head(p_fq));
        }
    }

    printf("%d:", *p_time);

    if (proc->data->ptimearrival < *p_time) {
        --proc->data->ptimeleft;
        ++proc_time;
        printf("%s:", proc->data->pname);
    } else {
        printf("idle:");
    }

    if (is_empty(&q)) {
        printf("empty:\n");
    } else {
        print_q(&q);
        printf(":\n");
        if (proc->data->ptimeleft == 0) {
            proc = remove_from_head(&q);
            proc_time = 0;
        }
    }

    if (proc_time == time_slice) {
        add_to_tail(&q, proc);
```

```
            proc = remove_from_head(&q);
            proc_time = 0;
        }
    }
    free(proc);
}
```

Explanation:

- It manages two queues: p_fq for processes awaiting execution and q for processes currently in execution.
- The code iterates in a loop, incrementing a time counter with each iteration.
- It checks conditions for exiting the loop: if p_fq is empty, the current process has no time left, and q is empty.
- If p_fq isn't empty and the next process is ready based on its arrival time, it is moved from p_fq to q.
- The code prints the current time and the state (executing or idle) of the processes, and updates the time remaining for the executing process.
- If a time slice is reached (defined as time_slice = 1), the currently executing process is moved back to the end of the queue q.
- If the executing process finishes, it is removed from q.
- The loop continues until scheduling is complete, and memory for proc is freed.

MakeFile:

Code:

```
CC = gcc
TRGT = scheduler
SRC = $(TRGT).c
CFLAGS = -Wall -Wextra

build:
    $(CC) $(CFLAGS) $(SRC) -o $(TRGT)

run:
    ./$(TRGT)

clean:
    rm -vf $(TRGT)
```

Explanation:

make: This command will execute the first target in the Makefile, which is build. It will compile the source code and generate the target executable.

make run: This command will execute the run target, which runs the compiled program.

make clean: This command will execute the clean target, which removes the target executable file. It's often used to clean up the project directory before committing changes or when you want to remove the compiled files.

Performance Metrics:

Round Robin:

Test Case 0:

Throughput = 0.176

Average turnaround time = 12.333

Average response time = 1.667

Test Case 2:

Throughput = 0.176

Average turnaround time = 12.333

Average response time = 1.667

Test Case 05:

Throughput = 0.150

Average turnaround time = 26.000

Average response time = 3.500

Test case 10:

Throughput = 0.105

Average turnaround time = 36.333

Average response time = 3.500

Test Case 13:

Throughput = 0.111

Average turnaround time = 49.625

Average response time = 4.500

First-Come-First-Serve (FIFO)

Test Case 0:

Throughput = 0.176

Average turnaround time = 10.333

Average response time = 5.667

Test Case 2:

Throughput = 0.176

Average turnaround time = 10.333

Average response time = 5.667

Test Case 05:

Throughput = 0.150

Average turnaround time = 19.167

Average response time = 13.667

Test case 10:

Throughput = 0.105

Average turnaround time = 27.667

Average response time = 19.333

Test Case 13:

Throughput = 0.111

Average turnaround time = 36.500

Average response time = 28.625

Shortest Job First (SJF):

Test Case 0:

Throughput = 0.176

Average turnaround time = 9.000

Average response time = 4.333

Test Case 2:

Throughput = 0.176

Average turnaround time = 9.000

Average response time = 4.333

Test Case 05:

Throughput = 0.150

Average turnaround time = 16.833

Average response time = 11.333

Test case 10:

Throughput = 0.105

Average turnaround time = 23.000

Average response time = 14.667

Test Case 13:

Throughput = 0.111

Average turnaround time = 27.375

Average response time = 19.500

Shortest Time-to-Complete First (STCF):

Test Case 0:

Throughput = 0.059

Average turnaround time = 27.000

Average response time = 0.000

Test Case 2:

Throughput = 0.059

Average turnaround time = 27.000

Average response time = 0.000

Test Case 05:

Throughput = 0.025

Average turnaround time = 101.000

Average response time = 0.000

Test case 10:

Throughput = 0.018

Average turnaround time = 136.000

Average response time = 0.000

Test Case 13:

Throughput = 0.014

Average turnaround time = 218.000

Average response time = 0.000