CPU Scheduling Lab

This lab project addresses the implementation of CPU-scheduling algorithms in an operating system.

Each process in an operating system is managed by using a data structure called the Process Control Block (PCB). A PCB contains the process ID, arrival timestamp, total burst time, execution start time, execution end time, remaining burst time and the priority of the process in the system. The PCB class is defined as follows and is accessible from the rest of the code in this lab project:

```
struct PCB {
    int process_id;
    int arrival_timestamp;
    Int total_bursttime;
    int execution_starttime;
    int execution_endtime;
    int remaining_bursttime;
    int process_priority;
}
```

The set of processes in the operating system that are ready to execute are maintained in a data structure called the Ready Queue. This data structure is an array of PCBs of the processes that are waiting for the CPU to become available so that they can execute.

The NULLPCB is defined as [PID:0, AT:0, TBT:0, EST:0, EET:0, RBT:0, Priority:0]

To determine the schedule of execution for the processes in an operating system, we consider three policies:

- Priority-based Preemptive Scheduling (PP)
- Shortest-Remaining-Time-Next Preemptive Scheduling (SRTP)
- Round-Robin Scheduling (RR)

In order to implement the above policies, we need to develop methods that handle arrival of processes for execution and the completion of process execution. Whenever a process arrives for execution, it can either join the ready queue and wait for its chance to execute or execute

immediately (if there is no other process currently executing or if the currently-running process can be preempted). Whenever a process completes execution, another process from the ready queue is chosen to execute next, based on the scheduling policy. The details of these methods are described below in the specification and you need to develop code for these functions that follow the specification and place the code in a file named **cpu.c**. You should include the oslabs.h file.

handle_process_arrival_pp:

This function implements the logic to handle the arrival of a new process in a Priority-based Preemptive Scheduler. Specifically, it takes five inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the PCB of the currently running process
4. the PCB of the newly arriving process
5. the current timestamp.

The method determines the process to execute next and returns its PCB.

If there is no currently-running process (i.e., the third argument is the NULLPCB), then the method returns the PCB of the newly-arriving process, indicating that it is the process to execute next. In this case, the PCB of the new process is modified so that the execution start time is set to the current timestamp, the execution end time is set to the sum of the current timestamp and the total burst time and the remaining burst time is set to the total burst time.

If there is a currently-running process, the method compares the priority of the newly-arriving process with the currently-running process. If the new process has equal or lower priority (smaller integers for the priority field in the PCB indicate higher priority), then its PCB is simply added to the ready queue and the return value is the PCB of the currently-running process. As the newly-arriving process is added to the ready queue, its execution start time and execution end time are set to 0, and the remaining burst time is the same as its total burst time.

If the new process has a higher priority, then the PCB of the currently-running process is added to the ready queue and the return value is the PCB of the new process. In this case, the PCB of the new process is modified so that the execution start time is set to the current timestamp, the execution end time is set to the sum of the current timestamp and the total burst time and the remaining burst time is set to the total burst time. Also, the PCB of the currently-running process is added to the ready queue after marking its execution end time as 0, and adjusting its remaining burst time.

The signature of the method is as follows:

```
struct PCB handle_process_arrival_pp(struct PCB ready_queue[QUEUEMAX], int
*queue_cnt, struct PCB current_process, struct PCB new_process, int timestamp);
```

A sample execution input and output:

input/output	parameter	value
input	ready_queue	EMPTY
input	queue_cnt	0

input	current_process	[PID:1, AT:1, TBT:4, EST:1, EET:5, RBT:4, Priority:8]
input	new_process	[PID:2, AT:2, TBT:3, EST:0, EET:0, RBT:3, Priority:6]
input	time_stamp	2
output	ready_queue	[PID:1, AT:1, TBT:4, EST:1, EET:0, RBT:3, Priority:8]
output	queue_cnt	1
output	PCB	[PID:2, AT:2, TBT:3, EST:2, EET:5, RBT:3, Priority:6]

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Priority-based algorithm.

handle_process_completion_pp:

This method implements the logic to handle the completion of execution of a process in a Priority-based Preemptive Scheduler. Specifically, it takes three inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the current timestamp.

The method determines the process to execute next and returns its PCB.

If the ready queue is empty, the method returns the NULLPCB, indicating that there is no process to execute. Otherwise, the method finds the PCB of the process in the ready queue with the highest priority (smaller integers for the priority field in the PCB mean higher priorities), removes this PCB from the ready queue and returns it. Before returning the PCB of the next process to execute, it is modified to set the execution start time as the current timestamp and the execution end time as the sum of the current timestamp and the remaining burst time.

The signature of the method is as follows:

```
struct PCB handle_process_completion_pp(struct PCB ready_queue[QUEUEMAX], int
*queue_cnt, int timestamp);
```

A sample execution input and output:

input/output	parameter	value
input	ready_queue	[PID:1, AT:1, TBT:4, EST:0, EET:0, RBT:4, Priority:23], [PID:2, AT:1, TBT:4, EST:0, EET:0, RBT:4, Priority:22], [PID:3, AT:1, TBT:4, EST:0, EET:0, RBT:4, Priority:24]
input	queue_cnt	3
input	time_stamp	2
output	ready_queue	[PID:1, AT:1, TBT:4, EST:0, EET:0, RBT:4, Priority:23], [PID:3, AT:1, TBT:4, EST:0, EET:0, RBT:4, Priority:24]
output	queue_cnt	2

output	PCB	[PID:2, AT:1, TBT:4, EST:2, EET:6, RBT:4, Priority:22]

Please refer to Section 2.5.3 of the Modern Operating Systems book for a detailed discussion of the Priority-based algorithm.

handle_process_arrival_srtp:

This method implements the logic to handle the arrival of a new process in a Shortest-Remaining-Time-Next Preemptive Scheduler. Specifically, it takes five inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the PCB of the currently-running process
4. the PCB of the newly-arriving process
5. the current timestamp. The method determines the process to execute next and returns its PCB.

If there is no currently-running process (i.e., the third argument is the NULLPCB), then the method returns the PCB of the newly-arriving process, indicating that it is the process to execute next. In this case, the PCB of the new process is modified so that the execution start time set to the current timestamp, the execution end time is set to the sum of the current timestamp and the total burst time and the remaining burst time is set to the total burst time.

If there is a currently-running process, the method compares the remaining burst time of the currently-running process and the total burst time of the newly-arriving process. If the new process does not have a shorter burst time, then its PCB is simply added to the ready queue and the return value is the PCB of the currently running process. As the newly-arriving process is added to the ready queue, its execution start time and execution end time are set to 0, and the remaining burst time is set to the total burst time.

If the new process has a shorter burst time, then the PCB of the currently-running process is added to the ready queue and the return value is the PCB of the new process. In this case, the PCB of the new process is modified so that the execution start time is set to the current timestamp, the execution end time is set to the sum of the current timestamp and the total burst time and the remaining burst time is set to the total burst time. Also, the PCB of the currently-running process is added to the ready queue, after marking its execution start time and execution end time as 0, and adjusting its remaining burst time.

The signature of the method is as follows:

```
struct PCB handle_process_arrival_srtp(struct PCB ready_queue[QUEUEMAX], int
*queue_cnt, struct PCB current_process, struct PCB new_process, int time_stamp);
```

A sample execution input and output:

input/output	parameter	value
input	ready_queue	EMPTY
input	queue_cnt	0

input	current_process	[PID:1, AT:1, TBT:8, EST:1, EET:9, RBT:8, Priority:0]
input	new_process	[PID:2, AT:2, TBT:6, EST:0, EET:0, RBT:6, Priority:0]
input	time_stamp	2
output	ready_queue	[PID:1, AT:1, TBT:8, EST:0, EET:0, RBT:7, Priority:0]
output	queue_cnt	1
output	PCB	[PID:2, AT:2, TBT:6, EST:2, EET:8, RBT:6, Priority:0]

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Shortest-Remaining-Time-Next algorithm.

handle_process_completion_srtp:

This method implements the logic to handle the completion of execution of a process in a Shortest-Remaining-Time Preemptive Scheduler. Specifically, it takes three inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the current timestamp.

The method determines the process to execute next and returns its PCB.

If the ready queue is empty, the method returns the NULLPCB, indicating that there is no process to execute next. Otherwise, the method finds the PCB of the process in the ready queue with the smallest remaining burst time, removes this PCB from the ready queue and returns it. Before returning the PCB of the next process to execute, it is modified to set the execution start time as the current timestamp and the execution end time as the sum of the current timestamp and the remaining burst time.

The signature of the method is as follows:

struct PCB handle_process_completion_srtp(struct PCB ready_queue[QUEUEMAX], int *queue_cnt, int timestamp);

A sample execution input and output:

input/output	parameter	value
input	ready_queue	[PID:1, AT:1, TBT:23, EST:0, EET:0, RBT:23, Priority:0], [PID:2, AT:1, TBT:22, EST:0, EET:0, RBT:22, Priority:0], [PID:3, AT:1, TBT:24, EST:0, EET:0, RBT:24, Priority:0]
input	queue_cnt	3
input	time_stamp	2
output	ready_queue	[PID:1, AT:1, TBT:23, EST:0, EET:0, RBT:23, Priority:0], [PID:3, AT:1, TBT:24, EST:0, EET:0, RBT:24, Priority:0]
output	queue_cnt	2

output	PCB	[PID:2, AT:1, TBT:22, EST:2, EET:24, RBT:22, Priority:0]

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Shortest-Remaining-Time-Next algorithm.

handle_process_arrival_rr:

This method implements the logic to handle the arrival of a new process in a Round-Robin Scheduler. Specifically, it takes six inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the PCB of the currently-running process
4. the PCB of the newly-arriving process
5. the current timestamp
6. the time quantum.

The method determines the process to execute next and returns its PCB.

If there is no currently-running process (i.e., the third argument is the NULLPCB), then the method returns the PCB of the newly-arriving process, indicating that it is the process to execute next. In this case, the PCB of the new process is modified so that the execution start time is set to the current timestamp, the execution end time is set to the sum of the current timestamp and the smaller of the time quantum and the total burst time. The remaining burst time is set to the total burst time.

If there is a currently-running process, the method simply adds the PCB of the newly-arriving process to the ready queue and the return value is the PCB of the currently running process. As the newly-arriving process is added to the ready queue, its execution start time and execution end time are set to 0, and the remaining burst time is set to the total burst time.

The signature of the method is as follows:

```
struct PCB handle_process_arrival_rr(struct PCB ready_queue[QUEUEMAX], int
*queue_cnt, struct PCB current_process, struct PCB new_process, int timestamp, int
time_quantum);
```

A sample execution input and output:

input/output	parameter	value
input	ready_queue	EMPTY
input	queue_cnt	0
input	current_process	[PID:1,AT:1,BT:8,EST:1,EET:9,RBT:8,Priority:0]
input	new_process	[PID:2, AT:2, TBT:8, EST:0, EET:0, RBT:8, Priority:0]

input	time_stamp	2
input	time_quantum	6
output	ready_queue	[PID:2,AT:2,TBT:8,EST:0,EET:0,RBT:8,Priority:0]
output	queue_cnt	1
output	PCB	[PID:1,AT:1,TBT:8,EST:1,EET:9,RBT:8,Priority:0]

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Round-Robin algorithm.

handle_process_completion_rr:

This method implements the logic to handle the completion of execution of a process in a Round-Robin Scheduler. Specifically, it takes four inputs:

1. the ready queue (an array of PCB structs)
2. The number of items in the ready queue
3. the current timestamp
4. the time quantum.

The method determines the process to execute next and returns its PCB.

If the ready queue is empty, the method returns the NULLPCB, indicating that there is no process to execute next. Otherwise, the method finds the PCB of the process in the ready queue with the earliest arrival time, removes this PCB from the ready queue and returns it. Before returning this PCB, it is modified to set the execution start time as the current timestamp and the execution end time as the sum of the current timestamp and the smaller of the time quantum and the remaining burst time.

The signature of the method is as follows:

struct PCB handle_process_completion_rr(struct PCB ready_queue[QUEUEMAX], int *queue_cnt, int time_stamp, int time_quantum);

A sample execution input and output:

input/output	parameter	value
input	ready_queue	[PID:1, AT:22, TBT:8, EST:0, EET:0, RBT:8, Priority:0], [PID:2, AT:21, TBT:8, EST:0, EET:0, RBT:8, Priority:0], [PID:3, AT:23, TBT:8, EST:0, EET:0, RBT:8, Priority:0]
input	queue_cnt	3
input	time_stamp	24
input	time_quantum	10
output	ready_queue	[PID:1, AT:22, TBT:8, EST:0, EET:0, RBT:8, Priority:0], [PID:3, AT:23, TBT:8, EST:0, EET:0, RBT:8, Priority:0]
output	queue_cnt	2

output	PCB	[PID:2, AT:21, TBT:8, EST:24, EET:32, RBT:8, Priority:0]

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Round-Robin algorithm.

This method is not implemented in Fall 2019 Test Suite

handle_end_of_time_quantum_rr:

This method implements the logic to handle the completion of a time quantum in a Round-Robin Scheduler. Specifically, it takes five inputs:

1. the ready queue (an array of PCB structs)
2. the number of items in the ready queue
3. the PCB of the currently running process
4. the current timestamp
5. the time quantum

The method determines the process to execute next and returns its PCB.

If there is a currently-running process (i.e., the third argument is not the NULLPCB), its PCB is added to the ready queue. Its execution start time and execution end time are set to 0 and its remaining burst time is reduced by the time quantum. The arrival time of its PCB is set to the current timestamp as if the process is just entering the ready queue now for the first time. The method then finds the PCB of the process in the ready queue with the earliest arrival time, removes this PCB from the ready queue and returns it. Before returning this PCB, it is modified to set the execution start time as the current timestamp and the execution end time as the sum of the current timestamp and the smaller of the time quantum and the remaining burst time.

The signature of the method is as follows:

struct PCB handle_end_of_time_quantum_rr(struct PCB ready_queue[QUEUEMAX], int *queue_cnt, struct PCB current_process, int time_stamp, int time_quantum);

A sample execution input and output:

input/output	parameter	value
input	ready_queue	[PID:1, AT:22, Priority:0, TBT:8, RBT:8, EST:0, EET:0], [PID:2, AT:21, Priority:0, TBT:8, RBT:8, EST:0, EET:0], [PID:3, AT:23, Priority:0, TBT:8, RBT:8, EST:0, EET:0]
input	queue_cnt	3
input	current_process	[PID:4,AT:21,Priority:0,BT:9,RBT:9,EST:21,EET:30]
input	time_stamp	24

input	time_quantum	10
output	ready_queue	[PID:1, AT:22, Priority:0, TBT:8, RBT:8, EST:0, EET:0], [PID:3, AT:23, Priority:0, TBT:8, RBT:8, EST:0, EET:0], [PID:4,AT:21,Priority:0,BT:9,RBT:-1,EST:0,EET:0]
output	queue_cnt	3
output	PCB	[PID:2, AT:21, Priority:0, TBT:8, RBT:8, EST:24, EET:32],

Please refer to Section 2.4 of the Modern Operating Systems book for a detailed discussion of the Round-Robin algorithm.