1 Scheduler

Scheduling is the process of determining the order of executing tasks or processes on the central processing unit (CPU) of a computer. The main objectives of scheduling are to optimize system resource usage, improve system performance, and provide a fair and efficient execution environment for multiple processes.

In this simple operating system, we use the Multilevel Queue scheduling algorithm, based on the scheduling algorithm in Linux.

The scheduler operates as follows: when a new program appears, the loader creates a new process and assigns a PCB (Process Control Block) to it. Then, the loader reads and copies the content of the program into the text segment of the new process. The PCB of the process is pushed into the ready queue list according to its priority. After that, the loader yields the CPU. The CPU executes processes using the Round Robin method. Each process is allowed to run for a defined time period. After that, the CPU must put this process back into the ready queue list. The CPU then selects another process from the ready queue list and continues execution.

In this system, we implement the Multi-Level Queue (MLQ) algorithm. The system includes MAX_PRIO priority levels. We simplify the process of adding to the queue and placing processes by putting them in the appropriate ready queue based on priority. The MLQ algorithm will use get_proc to retrieve a process and then give it to the CPU for processing.

Description of the MLQ algorithm: we set up the ready_queue_list based on priority (prio), where slot = (MAX_PRIO - prio). Each ready_queue has a fixed number of slots to use the CPU, and when these are used up, the system must transfer resources to other processes in the next ready_queue and leave the remaining work for the next slots in the future, even if it has to restart a cycle of the ready_queue_list.

Example: assume we have the following configuration:

- MAX_PRIO: 2
- num processes: 2
- num cpus: 1
- 1 time quantum = 1 slot

Process	Arrival time	Burst time	Priority
P1	0	4	1
P2	1	5	0

Then we have the Gantt chart as follows:

A	B	C	D	E	F	G	H	I	J	K	
	P1	P2	P2	P1	P2	P2	P1	P2	P1		
	0	1	2	3	4	5	6	7	8	9	timeslot

Figure 1: Gantt Chart of MLQ Scheduling Algorithm

Explanation: priority 0 has MAX_PRIO - prio = 2 - 0 = 2 slots, priority 1 has 1 slot. The CPU runs processes in Round Robin fashion, meaning that each process runs for a time period equal to the time quantum. For higher priorities, processes get more execution time, so queue 0 receives 2 slots for execution, with each slot running for a time quantum period.

1.1 Function Implementation

We will go through the files and the functions that need to be implemented, along with their meaning and how to implement them:

queue.c:

- **enqueue**: adds a PCB to the `ready_queue`, which is an array with size `MAX_QUEUE_SIZE`. In cases where `ready_queue` is full, we simply do not add. Otherwise, we add it to the end of the array and update the `size` variable.
- **dequeue**: removes a PCB from the `ready_queue` based on priority. We simply find the process with the highest priority (lowest priority value) in the queue. After finding that process, shift all processes behind it one position forward. Update the queue size (decrease size by 1). Return the highest priority process that was removed.

sched.c:

- **get_mlq_proc**: This function is used to fetch the next process in the `ready_queue_list`. Since it must maintain the state of which `ready_queue` is being used and how many `slot`s are left, we can use either a global variable or a static variable. Our group chose solution 2, and the implementation proceeds as follows:
  - Lock synchronization to prevent data races when multiple CPUs are involved.
  - If the current `ready_queue` has no more slots, move to the next non-empty `ready_queue` and update the `slot` variable.
  - Retrieve the PCB from the `ready_queue` and update `slot`.
  - Unlock and return the result.

1.2 Question

What is the advantage of using a priority queue compared to other scheduling algorithms?

Answer:

Compared to other scheduling algorithms (FCFS, SJF, SRTF, ...), priority queue scheduling in an operating system has the following advantages:

- **Efficient resource allocation:** Priority queue scheduling helps optimize CPU usage by allowing high-priority processes to be executed in time, while still allowing lower-priority processes to run, avoiding resource starvation.
- **Fairness:** Because the implementation is based on round-robin within each priority, every process has a chance to execute.
- **Support for real-time tasks:** Due to its fairness, this scheduling algorithm is effective in supporting real-time and short-deadline tasks.