Operating Systems 2018/19
Assignment 5

Prof. Dr. Michael Beigl
Mathias Gottschlag, M.Sc.
Jonas Kiemel, M.Sc.

Submission Deadline: Sunday, December 2nd, 2018 – 23:59

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

The assignments must be filled out online in ILIAS. Handwritten solutions are no longer accepted. You will find the online version for each assignment in your tutorial's directory. **P-Questions** are programming assignments. Download the provided template from ILIAS. Do not fiddle with the compiler flags. Submission instructions can be found on the first assignment.

In this assignment you will get familiar with scheduling basics and policies.

T-Question 5.1: Scheduling

a. Why are the scheduler and dispatcher good examples for the distinction between policy and mechanism? **1 T-pt**

b. Briefly explain the difference between cooperative and preemptive scheduling? What problem does preemptive scheduling solve? **2 T-pt**

c. With lottery scheduling, every process is assigned a certain number of tickets. To make a scheduling decision the lottery scheduler randomly chooses a ticket and selects the process that owns the ticket. Briefly explain how lottery scheduling can be implemented without allocating any dedicated objects per ticket such as structs, integers, or array elements. **2 T-pt**

d. Give the scheduling sequence (e.g., P_X, P_Y, P_Z,...) for the following processes with round robin scheduling and a timeslice length of 1 time unit. The scheduler first adds new processes (if any) to the tail of the ready queue and then inserts the previous process to the tail (if it is still runnable). **2 T-pt**

Process	Burst length	Arrival time
P_1	3	0
P_2	5	2
P_3	2	4

e. Calculate the average waiting time for the example in 5.1d. **1 T-pt**

P-Question 5.1: Priority Scheduler

Download the template **p1** for this assignment from ILIAS. You may only modify and upload the file `scheduler.c`.

Priority scheduling assigns each scheduling entity (i.e., a process or thread) a priority. For each priority the scheduler has a ready queue into which ready threads with the respective priority are enqueued. The scheduler always selects the first thread from the ready queue of the highest priority (here: small integer ≡ low priority). If multiple threads have the same priority (i.e., a queue contains more than one thread), the scheduler employs round robin scheduling within the queue.

Refer to the lecture slides for more details. In this question you will write your own priority scheduler.

a. The queue implementation in the template already contains the necessary structures to represent a queue (`Queue`) and its elements (`QueueItem`). The queue contains a `head` and a `tail` pointer to make it possible to reference both, the first and the last item, in O(1). Implement the functions to add and remove elements. You can use the following guideline: **2 P-pt**

_enqueue Adds a new item to the queue's *tail*.

- Allocates a new `QueueItem` with `malloc` (silently fail on errors)
- Assigns the supplied `data` to the new item
- Adds the new item to the tail of the queue by updating the `head` (if necessary) and `tail` pointers as well as the `next` pointer of the current tail element (if any)

_dequeue Removes an item from the queue's *head*.

- Returns -1 if the queue is empty
- Otherwise, removes the first item from the queue's head by updating all necessary pointers
- Frees the item with `free`
- Returns the `data` field of the removed item (Caution: Remember that you cannot access the item anymore after freeing it!)

Hints: If the queue is empty `head` should be `NULL`. You may also set the `next` pointer of the last element to `NULL`.

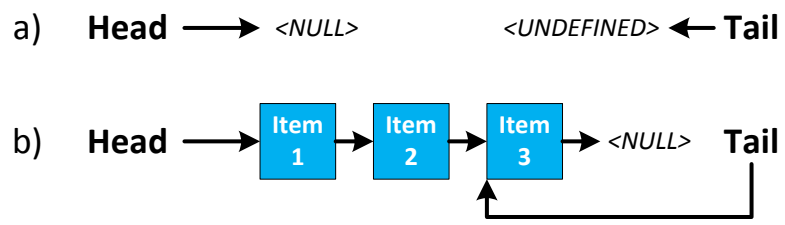

Abbildung 1: Example queue. a) Empty queue b) Queue with 3 items

```
void _enqueue(Queue *queue, int data);
int _dequeue(Queue *queue);
```

Note to a): It is important that you are able to solve this type of straight C programming questions without a big hassle. Otherwise, you won't be able to get OS related programming tasks to fly in an appropriate time to master the certificate exam. If you have a hard time getting the queue to work, consider investing some time reading one of the many good books on the C programming language (e.g., Brian W. Kernigham and Dennis M Ritchie *The C Programming Language*). You won't regret it!

b. Implement the event handler functions, which set the supplied thread's state and if necessary add the thread to the appropriate ready queue. Set the `QueueItem`'s data field to the thread's id.

Hints: Take a look at the lecture and the comments in the template for details an thread state changes. **2 P-pt**

```
void onThreadReady(int threadId);
void onThreadPreempted(int threadId);
void onThreadWaiting(int threadId);
```

c. Implement the priority scheduling policy with prevention of starvation. Your scheduling function should fulfill the following requirements: **4 P-pt**

- Find the ready queue with the highest priority that contains a ready thread
- Removes the first thread from the queue, updates its state and returns its thread id
- Applies the following starvation prevention:
  If the scheduler selects threads with a certain priority more then 5 times without selecting a thread with a lower priority, the scheduler resorts to the next lower priority queue that (1) is not empty and (2) does not break this starvation rule (i.e., selected more than 5 times whereas no threads with even lower priority were scheduled inbetween). If no such queue is available, the scheduler behaves as if a thread with low priority had been scheduled, resets the queues' counters accordingly, and selects a thread from the non-empty queue with lowest priority.

Hints: The simplest solution to the starvation prevention will use a recursive approach to determine the right queue for thread selection.

```
int scheduleNextThread();
```

Total:
8 T-pt
8 P-pt