VE482 — Introduction to Operating Systems

Homework 5

Manuel — UM-JI (Fall 2019)

Non-programming exercises:
- Write in a neat and legible handwriting
- Clearly explain the reasoning process
- Write in a complete style (subject, verb and object)

Progamming exercises:
- Write a README file for each program
- Upload an archive with all the programs onto Canvas

Ex. 1 — *Simple questions*

1. A system has two processes and three identical resources. Each process needs a maximum of two resources. Can a deadlock occur? Explain.
2. A computer has six tape drives, with n processes competing for them. Each process may need two drives. For which values of n is the system deadlock free?
3. A real-time system has four periodic events with periods of 50, 100, 200, and 250 msec each. Suppose the four events require 35, 20, 10, and x msec of CPU time, respectively. What is the largest value x for which the system is schedulable?
4. Round-robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred more than once in the list? Would there be any reason for allowing this?
5. Can a measure of whether a process is likely to be CPU bound or I/O bound be detected by analyzing the source code. How to determine it at runtime?

Ex. 2 — *Deadlocks*

Assuming three resources consider the following snapshot of a system.

Process	Allocated	Maximum	Available
P_1	010	753	332
P_2	200	322	
P_3	302	902	
P_4	211	222	
P_5	002	433	

1. Determine the content of the Request matrix.
2. Is the system in a safe state?
3. Can all the processes be completed without the system being in an unsafe state at any stage?

Ex. 3 — *Programming*

Implement the Banker's algorithm.

Ex. 4 — *Minix 3*

How is scheduling handled in Minix 3? Provide clear explanations on how to find the information just by exploring the source code of Minix kernel.

Ex. 5 — *The reader-writer problem*

In the *reader-writer problem*, some data could be accessed for reading but also sometimes for writing. When processes want to read the data they get a *read lock* and a *write lock* for writing. Multiple processes could get a read lock at the same time while a write lock should prevent anybody else from reading or writing the data until the write lock is released.

To solve the problem we decide to use a global variable `count` together with two semaphores: `count_lock` for locking the `count` variable, and `db_lock` for locking the database. To get a write lock we can proceed as follows:

```
void write_lock() {
  down(db_lock);
}
```

```
void write_unlock() {
  up(db_lock);
}
```

1. Explain how to get a read lock, and write the corresponding pseudocode.
2. Describe what is happening if many readers request a lock.

To overcome the previous problem we will block any new reader when a writer becomes available.

3. Explain how to implement this idea using another semaphore called `read_lock`.
4. Is this solution giving any unfair priority to the writer or the reader? Can the problem be considered as solved?