laSalle
UNIVERSITAT RAMON LLULL

Operating Systems 2021-2022

Session 8: Mutual Exclusion 02/12/21

Goals

In previous sessions we have worked on the concept of threads and forks for parallelism and concurrency of tasks. The problem with working with shared resources and multi-process systems is that two processes try to access the same resource simultaneously, thus generating unpredictable or random results. On the other hand, sometimes the correct sequencing of tasks (synchronization) must also be guaranteed. The goal of this session is to introduce Operating Systems students to one of the most common synchronization and mutual exclusion mechanisms: **semaphores**. With the development of this session it is intended that the student clarify the theoretical concepts of the use of semaphores and solve different problems.

Motivation

More specifically, with this session, the student will exercise:

- Semaphore creation (*SEM_constructor, SEM_init*)
- Control and destruction of semaphores (*SEM_destructor*)
- Operations on semaphores (*SEM_signal, SEM_wait*)
- Our own library semaphore.h
- Use of the mutex library to use semaphores with threads.

Previous documentation

To complete this session, it is recommended to read of the following references:

SALVADOR, J. (2014). *Programació en UNIX per a pràctiques de Sistemes Operatius*.

STEVENS, W. R. & RAGO S.A. (2008). Advanced Programming in the UNIX Environment, 2nd edition.

TANENBAUM, A. S. (2009). Modern Operating Systems, 3rd Edition.

Dwarfs

In her day to day, to avoid jealousy among the dwarfs, Snow White forces them to make their own food.

Since the last dinner, Snow White, tired of the dwarfs fighting, has contacted the software department of La Salle University, to ask us if we can solve the problem she has with her dinner shifts. From the Software Department, it has been decided that the students will solve the problem for Snow White.

For that, the task of LaSalle students will be to program a **main process that launches N threads of execution (forks)**, where each thread represents a dwarf.

Although it would be fun to watch, we do not want the dwarfs to fight, therefore each dwarf will have to wait his turn to use the oven, the stove, the fridge and finally clean up everything.

To perform the testing of the program, we will have a text file with the time it takes each N dwarf to perform each action. The file (called dwarfs.txt) will have the following structure:

dwarfs.txt

```
3
F3-O9-S3-C4
F5-O5-C8
F4-O8-S3-C9
```

The first value (in this case 3) is the number of dwarfs in the house that day. Since each dwarf does not take the same time to perform the tasks, F3 represents the time in seconds that he will be taking things from the fridge, O9 the time he is using the oven, S3 and C4, the time on the stove and the time to clean up.

As we can see, all the dwarfs **do not have to do the four tasks**, in this case the second dwarf will not use the stove, but they will always follow the sequence in the same order.

For the design of the program, we will take into account that in the kitchen up to 2 dwarfs can use the oven at the same time, 4 the stove, 1 the fridge and 2 dwarfs can clean up the plates at the same time. When finishing all his tasks, the dwarf goes to rest.

Execution example:

```
montserrat:~/ > s7
Dwarfs ready!
Dwarf 0 ready to cook!!
Dwarf 1 ready to cook!!
Dwarf 1 in the fridge!!
Dwarf 2 ready to cook!!
Dwarf 1 in the oven!!
Dwarf 0 in the fridge!!
Dwarf 0 in the oven!!
Dwarf 2 in the fridge!!
Dwarf 1 cleaning!!
Dwarf 2 in the oven!!
Dwarf 0 in the stove!!
Dwarf 0 cleaning!!
Dwarf 2 in the stove!!
Dwarf 2 cleaning!!
All dwarfs are resting!
```

Considerations

- The text file will be properly formatted and filled. The letters will always be in uppercase and the numbers will always be a single digit.
- The use of global variables is not allowed, except those that you think are essential.
- All processes must be created using forks.
- Recursively programming is not allowed. All software has to be iterative.
- Check not to leave shared resources (semaphores, zombie processes, etc.) in the system. Remember to use ipcs, ipcrm and the provided scripts. And freeing all the dynamic memory.
- It must be ensured that at the end of the execution all the processes have ended correctly without leaving zombie or orphaned processes.
- All inputs and outputs must be done using file descriptors. **The use of printf, scanf, FILE*, getchar or similar is NOT allowed.**
- **The use of the functions "system", "popen", or from the same family is NOT allowed**.
- **You must compile using the –Wall, –Wextra and –lpthread flags.**
- **Any deliverable containing warnings or errors will be directly discarded.**
- At the start of the .c you must include a comment with your logins, names, and surnames.
- **For submitting the session, you must hand over a file "S8.c", and delivered it through eStudy platform.**