Operating Systems

Lab 9 Exercise – Processes, threads, and critical regions

Exercise 1

Implement a C program, **`generation_tree_number.c`**, which receives a command line parameter: **`n`**. The parent **process**, pushes its ID number (**`1`**) in a vector, creates two children and terminates. Each child, pushes its number in its vector, creates another two children, and terminates.

Process creation stops after 2^n leave processes have been created.

Each leaf child pushes its number in its vector, and print its generation tree, i.e., the sequence of its saved IDs. Example:

```
> generation_tree_number 3
Process tree: 1 3 7 15
Process tree: 1 2 5 11
Process tree: 1 3 7 14
Process tree: 1 2 5 10
Process tree: 1 3 6 13
Process tree: 1 2 4 9
Process tree: 1 3 6 12
Process tree: 1 2 4 8
```

Hint:
Draw the tree for n=3 and check that the number of the process at the second level begins by 2, the number of the process at the third level begins by 4, number of the process at the fourth level begins by 8 etc..)

Exercise 2

Implement a C program, **`generation_threads_tree_number.c`**, which behaves similarly to **`generation_tree_number.c`**, but prints the sequence of thread numbers.
Please notice that **we assume that we cannot pass variables** in **`pthread_create()`**.
Thus, each thread gets its ID number from a global variable initialized to **`1`**.

Exercise 3

Implement a C program that simulates two clients that perform concurrent bank transactions on accounts **`1`** and **`2`**.
The main thread creates two threads, which are owners of both accounts.
Initially the total money in the two accounts is 100000 Euro.
Thread **`i`** loops sleeping for a random interval (**`0-8`**) of seconds, then it reads its current balance in account **`i`**, and after a random interval (**`0-4`**) of seconds, it decides to move a random amount of money **`m,`** (**`1000-5000`**) Euro, from account **`i`** to account **`3-i`**.
Of course, the transaction implies that the balance of account **`i`** decreases by **`m`**, and the balance of account **`3-i`** increases by **`m`**, the read and write operations on the same account must be performed in mutual exclusion.
Write a single code for the two threads.
Analyze your solution, and comment the possibility of deadlock.