P5: Final Review

Sergio López and Xavier Marquès

March 2021

1. **01-Process-ExecCmds.c**
   When we apply execvp, it replace the current process with another that execute, in this case, the first command, so the old process is killed and the other command is not executed. If we want to execute all commands, we need to create two process with fork() and in each child apply the execvp().

2. **02-Process-Sequential.c**
   To execute the process sequentially we need to include the wait inside the for. This way, in each iteration the program creates a new process and wait to finish producing a sequential schema.

3. **03-Threads-NoThreads.c**
   So we can delete all parts regarding the threads and we can call the function directly inside the for loop passing the same argument, so we're getting the same result.

4. **04-Threads-Join.c**
   The problem is that the thread creation and execution is asynchronous and since we're passing a memory address, this can be changed in the meantime or two or more threads can get the same value at same time, producing this printing problems with the id. To fix that, we need to create a variable using dynamic memory and pass the value by reference.

5. **05-Synch-Sum.c**
   We need to create 4 locks, 1 per position of the array, initialize them with a for loop and, in the thread function where is the critical part, apply this lock / unlock . The code we added:

```

    pthread_mutex_t locks[NSUMS]; //create locks

    // **** inside the thread function ****
    pthread_mutex_lock(&locks[i%NSUMS]);
    //critical part
    pthread_mutex_unlock(&locks[i%NSUMS]);

    //in the main, initializing
    for (int i=0; i<NSUMS; i++){pthread_mutex_init(&locks[i],
    NULL);}

```

6. **06-Comm-Pipe.c**
   First we declare the file descriptor, then initialize the pipe. Since in the child we want to read from the parent, we need to first close the writing part of the pipe and then apply the read. In the parent, the other way around , we close the reading part and apply the write. We've created a new int variable to see if the pipes are working, so the parent pass the i value to the pipe and the child takes this value from the pipe and stores to this new variable, and then we print it.

7. **07-Comm-FileSem.c**
   We need to create four semaphores instead of one. So we add this code in the main to do so:

```
  sem_t* mutex[]= {sem_open("mutex1", O_CREAT, 0600, 1),
      sem_open("mutex2", O_CREAT, 0600, 1),
      sem_open("mutex3", O_CREAT, 0600, 1),
      sem_open("mutex4", O_CREAT, 0600, 1)};

```

   Inside the child, we add again the previous code to open the semaphores inside and then, in the critical section we need to use the atomic operations of wait and post:

```
for(int i=0;i<500;i++) {
        sem_wait(mutex[i%NSUMS]);
        // **** CRITICAL SECTION ****
        sem_post(mutex[i%NSUMS]);
    }

```

   Finally, is important to close the semaphores at the end:

```
  for(int i=0;i<NSUMS;i++){sem_close(mutex[i]);}

```