University of Liège

Project 1 : OShell

Operating systems

Maxime Meurisse
Valentin Vermeylen

Master in Engineering and Computer Science
Academic year 2019-2020

1 Processes management

To manage sequential process execution, we have implemented a loop that iterates n times (n being the number of executions of the command). At each iteration of this loop, we create a new process with `fork`. The child process executes the command with `execvp` or terminates in case of a problem. The parent process waits for the child process to finish executing (a 5 second timeout is implemented). When the child process is finished, the command is added to the command list (by the parent) and the next iteration of the loop can begin.

To manage parallel process execution, we have implemented two loops that iterate each n times (n being the number of times the command is executed). The first loop will create a new process at each iteration with `fork`. Each child process executes the command with `execvp` or terminates in case of a problem. The second loop is a series of n `wait` allowing the parent to wait for the end of each child's execution.

The major difference between sequential and parallel execution is that in the first case, a `fork` is directly followed by a `wait`, whereas in the second case, all the `fork` are executed first and the `wait` come afterwards.

2 Process list management

To memorize the list of commands performed, we have implemented a simplified version of a vector (acting as a simple stack). Its starting capacity is 10. When the vector is full, its size is doubled (dynamically with `malloc` and `realloc`). Each element of the vector is a structure containing the command performed, the PID of the process that performed the command, and the exit code of the process.

To export the contents of the vector into a `memdump.bin` file, we first write an integer representing the number of elements contained in the vector. We then write the information composing each element of the vector sequentially. Each command is preceded by its size so that when reading the file, a `char*` of the right size can be allocated.

For example, if the list is as follows :

```
(ls;10;0)->(uname;11;0)
```

the content of the file[1] will be

```
22ls1005uname110
```

3 Time spent on the project

The project was achievable in a reasonable amount of time. The design of the OShell itself was completed fairly quickly (5 hours at most).

We spent most of the time debugging the program because we hadn't done C for a while and some points had escaped us (memory allocations, etc).

In total, we spent about 15 hours on this project.

[1] The content of the file is written legibly for this example. In practice, the content is binary and therefore unreadable for a human reader.