2543289, sbn338
Bas van der Borden

Internet Programming

Assignment 1

All C programs have been tested on Ubuntu 16.04 LTS. The Java programs have been tested with JDK 8, on Mac OSX 10.13.6. This has been done because unfortunately I ran into multiple problems when trying to install the JDK on Ubuntu.

The first shells, mysh1 through mysh4 have been implemented in such a way that they are invoked by simply running ./mysh1 after running the makefile. After this the shells will print out the dollar sign and wait for the command to be processed.

For the first shell I chose to use a while(1) loop to continuously keep the program running. The input is then gathered through the fgets command. The input is then stripped from the newline character and divided in a program and path variable. The program variable is then used to determine the task, the path variable is only used when changing directory.

The second shell extends the first shell with support for arguments with the programs to execute. To support this I used the execvp command instead of the execlp command in the first shell. Furthermore to support the arguments a loop is added to separate the different arguments and place them in a array.

mysh3 adds support for a single piped commands to the previous shell. When the program requested is not cd or exit the shell will create a pipe. The forked process will then write to … , close the pipes and run the requested program. If a piped command is requested a second forked process will be created which reads from the pipe, closes the pipes and executes the command. After this the parent process will close the pipes again.

The last shell was difficult to implement and took a lot of iterations to get right. In the end I chose to create a separate void for the execution of the command through the fork call. In addition to the previous shell this script takes into account the memory allocation for the pipes. The input is split according to the amount of pipes needed and thus the execution is dependant on the number of pipes. This is handled through a simple for loop with if statements for the first, last and other commands, because the pipes need to be handled different in these three situations. Furthermore the new void function only closes the pipes it uses.

The synchronisation processes are all called in a similar manner, e.g. ./syn_process_1 after which the output is shown. For the synchronisation tasks the first process was modified with semaphores. A semaphore is created at the start and then before the two display calls a down is performed. Since this would put both processes in the wait state, the parent process first performs a up on the semaphore before entering the for loop.

The second process uses the same implementation of the semaphore as this already gives the correct output for the task. I chose to use the semaphores in the first process since I already had some experience with them and found them the easiest to use.

Then the two synchronisation processes were implemented with the use of threads. The two implementations use the same display file as the previous implementation. The thread implementations use mutex for the synchronisation. The first implementation creates two threads which call the function Print with the arguments "Hello world" and "Bonjour monde". This Print function then uses the mutex lock to make sure only one thread can execute display at a time.

The second implementation also uses the mutex but is extended with the condition variables. For this the Print function is replaced with two different functions, one for each string to be printed. Through the use of the conditional variable we get our correct output.

To build the java versions run the included build.sh file. The first java implementation can be called with "java Main", the second implementation with "java Main2".

The java version of the first synchronisation task consists of three different classes. The Display class is a as plain as possible conversion of the display function given for the c part. The program is run by calling the Main class, this creates the display and two threads and starts both of them. The MyThread class uses the arguments with which the threads are called to perform either "hello world" or "bonjour monde". Before either of them is called we first look at the synchronised statements, which gives us the wanted result.

The second process, Syn2, extends the first process with a class Print. This class is called from MyThread instead of the Display. By doing so we can implement a boolean lock and thus the wait() and notifyAll() functions. To make sure that “ab” is always before “cd\n” it only waits when the boolean lock is false whilst “cd\n” waits when lock is true. After they have called display to print the string, they set the boolean and notifyAll() to wake all processes.