Internet Programming

Programming Assignment 1: Unix Multiprocessing

Deadline: Sunday 20 September 2015, midnight

1 Writing a Micro-Shell

A shell is a program which reads commands from the keyboard and creates the appropriate processes to execute them. Whenever you type a command in a Unix system, the program which reads and starts your command is a shell.

Usually, shells have a multitude of additional functions, such as interpreting scripts, manipulating environment variables, and so on. The goal of this exercise is to write a minimalistic shell, which only reads commands from the keyboard and executes them.

1.1 Programming the shell

For the programming part of this assignment, you will have to implement three progressively more complex versions of a basic shell. These shell variants must be named `mysh1`, `mysh2`, `mysh3`. Following are the descriptions of the shell variants.

1. `mysh1`, which reads a program name from the keyboard. When a program name is read, the shell creates a new process to execute the requested program. The shell waits for the new process to terminate before accepting another command. For example, entering "`ls`" to your program should list the content of the current directory. Entering "`exit`" should terminate the shell. To indicate that the shell is currently idle and waiting for input, a "`$`" prompt must be displayed.

2. `mysh2` is an extension of `mysh1`. In addition to executing the specified program, it must also accept a number of parameters which will be supplied to the program. For example, your new shell should interpret correctly commands such as "`ls -l /tmp`".

3. `mysh3` is an extension of `mysh2`. It adds support for piped commands such as "`ls /tmp | wc -l`". You can assume that typed commands will contain at most one pipe. I.e., you do not need to support "chained pipes" like "`sort foo | uniq -c | wc -l`".

   
   In the implementation of `mysh3`, you will need to use the `dup(2)` or `dup2(2)` functions. Have a look at their `man` pages. For parsing the command line, function `strtok(3)` may come in handy.

4. **Optionally** you may add the `cd` command to `mysh3`. The command must support both absolute (e.g., `cd /home/user`) and relative paths (e.g., `cd ../Pictures`). For implementing the `cd` command, you should look at the `man` pages for `chdir(2)` and `getcwd(3)`.


For this assignment, the use of `system()` function is forbidden. It is also forbidden to invoke another shell (e.g., `/bin/sh`) to do the work for you.

Make sure that any abnormal user input is handled gracefully. I.e., in the presence of abnormal user input, your shell must not abort.

1.2 Questions

Q-A How many processes must your shell create when receiving a piped command? How many pipes? Until when must the shell wait to accept another command?

Q-B Can you implement a shell program which only utilizes *threads* (instead of *processes*)? If your answer is yes, then write a thread-based version of `mysh1`, call it `mysh1_th`, and include it in your submission. If your answer is no, explain why.

Q-C Can you use the `cd` command with shell `mysh3`? Why?

2 Synchronization

Study the source code of program `syn1` (in Figure 1) and `syn2` (in Figure 2). The goal of this exercise is to use synchronization primitives so that the two programs produce the desired output **without changing the `display()` function**.

- For `syn1`, we want to prevent the two displayed messages from interpenetrating each other. E.g., this output is correct:

```
Hello world
Hello world
Bonjour monde
Hello world
Bonjour monde
Bonjour monde
```

  But this output is not correct:

```
HelBonlo world!
jour monde
HBeonljloo ur mwonordeld
```

- For `syn2`, we want the following output to be produced:

```
abcd
abcd
abcd
abcd
abcd
...
```

  Something like this is considered incorrect output:

```
abcabcd
d
abcabd
abccd
cabd
abcdab
cabd
abcdab
cabd
cd
```

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

void display(char *str) {
  char *tmp;
  for (tmp=str;*tmp;tmp++) {
    write(1,tmp,1);
    usleep(100);
  }
}

int main() {
  int i;

  if (fork()) {
    for (i=0;i<10;i++) display("Hello world\n");
    wait(NULL);
  }
  else {
    for (i=0;i<10;i++) display("Bonjour monde\n");
  }
  return 0;
}
```

Figure 1: Program `syn1`.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

void display(char *str) {
  char *tmp;
  for (tmp=str;*tmp;tmp++) {
    write(1,tmp,1);
    usleep(100);
  }
}

int main() {
  int i;

  if (fork()) {
    for (i=0;i<10;i++) display("ab");
    wait(NULL);
  }
  else {
    for (i=0;i<10;i++) display("cd\n");
  }
  return 0;
}
```

Figure 2: Program `syn2`.

2.1 Programming with Synchronization Primitives

2.1.1 Process Synchronization (`syn1`)

Modify the original source of `syn1` so that the desired output is produced.

Before starting coding, think which is the proper synchronization primitive that will help you achieve the desired behavior from the forked processes. Also think for which part(s) of the program you should enforce synchronization.

2.1.2 Process Synchronization (`syn2`)

Modify the original source of `syn2` so that the desired output is produced.

You may need to use a different synchronization primitive this time. Repeat the problem analysis process described above before going on with any implementation.

2.1.3 Thread Synchronization

Transform the programs `syn1` and `syn2` to use pthreads instead of processes. Use thread synchronization primitives as well. Call the new programs `synthread1` and `synthread2`.

2.1.4 Java Threads

Write the same two programs in Java (using Java threads). Call their source files `syn1.java` and `syn2.java`.

A Submission and Grading

A.1 Grading

Your final grade for each assignment will range from 0 to 100. Most assignments will include *optional sub-components* that make the assignment sum-up to a figure larger than 100. Solving the optional parts will only count towards compensating for any other errors. I.e., you will not get a 120/100 grade if you solve everything without errors.

Grading for the different parts of this assignment is as following:

Item	Points	Note
`mysh1`	6	
`mysh2`	6	
`mysh3`	12	
`cd` command for `mysh3`	5	optional
`mysh1_th`	10	optional
`syn1`	8	
`syn2`	8	
`synthread1`	12	
`synthread2`	12	
`syn1` (Java)	12	
`syn2` (Java)	12	
Questions	12	
Total	**100+15**	

A.2 What to submit

You must create and submit a zip file (name doesn't matter) with your assignment. Unless otherwise required, **all files must be stored on the top directory** of the archive. Make sure you don't include un-needed files in your submission (object files, executables, class files etc.).

You can use the following commands to create the zip file:

```
cd assignment1
zip -r ~/assignment1.zip *
```

The contents of the zip file should be as following:

- `Makefile` → A makefile containing rules that compile all the submitted programs. A `build` recipe must be provided that compiles everything. I.e., issuing the following command must result in all the required binaries being compiled:

  ```
  make build
  ```

- `mysh1.c`, `mysh2.c`, `mysh3.c` → The programs described in Section 1.1.
- `mysh1_th.c` → Optional. A possible proof implementation for the threaded version of `mysh1` (see **Q-B**).
- `syn1.c`, `syn2.c`, `synthread1.c`, `synthread2.c` → The programs described in Section 2.1.
- `answers.txt` or `answers.pdf` → Your answers to the questions in this assignment in **plain text** or **pdf** format. You must clearly indicate which answer corresponds to which question.

A.3 How to submit

To submit the zip file with your assignment, you should go to the following url.

https://checker.labs.vu.nl/ui/#IP

The submission website uses a self-signed certificate, so you should bypass the warning presented by your browser and accept the certificate. You will need a username and a password to log in to the website. If you don't have them, please contact the instructor and/or the teaching assistant ASAP. For groups consisting of two students, any of them may login to make the submission, which will of course count for both of them.

The programming parts of the assignment are evaluated automatically and you will get direct feedback if any problems are detected with your submission. After a successful upload, click "View Results" to see your submission being evaluated and get feedback for potential problems. You need to wait and click "View Results" again to see the results of the evaluation.

A.4 Disclaimers

Make sure you follow the archive structure described above in A.2. Otherwise, automatic evaluation will not work and you will end up with a null grade.

You may submit your assignment multiple times. **Only the last submitted version will be taken into account for your grade.**

Passing all the automated tests does not mean your submission is perfect! **Submissions may also be subject to manual inspection.**