Internet Programming

Programming Assignment 1: Unix Multiprocessing

Deadline: Friday 21 September 2012, 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.

Attention: For all this exercise, the use of the `system()` function is forbidden. It is also forbidden to call another shell (such as `/bin/sh`) to do the work for you...

1. Start by writing a program called `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.

2. Write program `mysh2`, an extension of `mysh1` that accepts a number of parameters in addition to the program name. For example, your new shell should interpret correctly commands such as "`ls -l /tmp`".

3. Now write program `mysh3`, an extension of `mysh2` that also accepts piped commands, such as "`ls /tmp | wc -l`". You can assume that typed commands will contain at most one pipe: for example, you do not need to support "chained pipes" like "`sort foo | uniq -c | wc -l`".

   Question 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?

   Hint: For this exercise, you will need to use the `dup` or `dup2` functions. Have a look at their `man` pages. For parsing the command line, function `strtok` may come in handy.

Question 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`. If your answer is no, explain why.

Question C: Can you use the `cd` command with your shell? Why?

Optional: Add the `cd` command to your shell. To do this, have a look at the `man` pages for `chdir()` and `getcwd()`.

2 Synchronization

(a) What does the following program do?

```
#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;
}
```

The goal of this exercise is to prevent the two messages from interpenetrating each other (without changing the `display()` function!). For example, 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
```

Question D: Which type of synchronization is required? Why? Which synchronization primitives must be used? Why? How must you use them?

Call this program `syn1`.

(b) What does the following program do?

```
#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;
}
```

The goal of this exercise is to make sure the output is *exactly* the following:

```
abcd
abcd
abcd
```

```
abcd
abcd
...
```

and not something like:

```
acbd
abcdab
cabd
abcdab
cabd
abcdab
cabd
cd
cd
cd
```

Again, changing the `display` function is forbidden.

Question E: What is the difference with the previous exercise? Which type of synchronization is required? Why? Which synchronization primitives must be used? Why? How?

Call this program `syn2`.

(c) 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`.

(d) Write the same two programs in Java (using Java threads). Call them `syn1.java` and `syn2.java`.

What to submit: All programs you wrote (mysh1, mysh2, mysh3, maybe a program for Question A depending on your answer, syn1, syn2, synthread1, synthread2, syn1.java, syn2.java). Do not forget including the Makefile! We should type "make" and everything should be compiled in one go. Also include documentation answering the questions, clearly indicating which answer corresponds to which question.

— end —