Unix Processing Assignment

Thomas Veerman and Martijn Vermaat
Department of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
{tveerman,mvermaat}@cs.vu.nl

February 28, 2007

1 Micro-shell

Question 1 Our shell has to create two child processes after receiving a piped command, one process for the writing command and one process for the reading command. One pipe has to be created for these processes to communicate (only in one direction). The parent shell process waits for both child processes to finish before reading the next command.

Question 2 This is not possible, because the necessary exec() call will destroy the entire shell process (including any threads).

Question 3 You cannot use the cd command in our shell. The cd commands in shells are built-in functions of these shells and not executable files.

2 Synchronization

Question 4 Mutual exclusion is needed here, because only one process at a time may be calling the display() function. Mutual exclusion can be implemented with a semaphore by initializing the semaphore to 1 and trying to DOWN() the semaphore before entering the mutual exclusive code and do an UP() afterwards. This way, only one process can be executing the mutual exclusive code at a time because DOWN() on a semaphore with value 0 will be blocking until UP() is called again.

```
DOWN()
mutex_code()
UP()
```

Question 5 The difference with the previous exercise is that the two mutual exclusive code blocks have to be executed alternatingly. In this case, events can be used by a process to wait for authorization and to authorize the other process to continue. Again, this can be implemented using semaphores, but this time we need two of them. The first semaphore implements the event that the first process may continue, while the second semaphore implements the event that the second process may continue. Each process authorizes the other process to continue execution by sending the appropriate event after it has executed the relevant part of his code.

```
DOWN(my_event)
mutex_code()
UP(his_event)
```

3 pthreads

Question 6 We could also have used semaphores in the pthread version, just like we did in the version with multiple processes. However, it would be a strange thing to do: allocate shared memory for a semaphore while the two threads share most of their variables anyway.

4 Java threads

Question 7 Yes. With semaphores one can do mutual exclusion, but also wait for events. In the Java version of the programs we could have simulated syn2.c by using two Semaphore objects, initialized to zero permits, and use *aquire()* and *release()* as DOWN() and UP() respectively. Both threads should be given a reference to these objects so that they both have access to them.