Lab Operating Systems
Session 2: Implementation of a shell + Memory management

1 Implementing your own shell – iwish

In this exercise, you will build your own (simple) command shell. The shell must at a minimum support the following features:

- start program which can be found in the user's search path (`$PATH`).
- I/O redirection, for example: `a.out < in > out`
- Background processes, for example: `a.out &`
- Pipes, for example: `a.out | b.out | c.out`

You can develop your lab on any UNIX system (Linux, BSD, MacOSX, ...). However, if you develop on a non-Linux system, please check that your program is compilable/testable on a Linux system before you submit your code, because the teaching assistant will use a Linux system to test your solution.
As a starting point you can use the skeleton program from the Minix-book:

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
      type_prompt( );                           /* display prompt on the screen */
      read_command(command, parameters);        /* read input from terminal */

      if (fork( ) != 0) {                       /* fork off child process */
          /* Parent code. */
          waitpid(−1, &status, 0);              /* wait for child to exit */
      } else {
          /* Child code. */
          execve(command, parameters, 0);       /* execute command */
      }
}
```

Note that the shell is a user level process, just like any other process (like an editor or a compiler). Therefore, you do not need to change anything in the OS its kernel (in fact, you are not allowed to). You will at least need some system calls, though. In particular, you will surely need the calls `fork`, `exec`, `open`, `close` and `dup`. Note that you need to spend extra care on termination of background process, in order to not flood the process table with zombie (defunct) processes.

Apart from the code, you are requested to also submit a report in which you briefly describe your design decisions, and the implementation of your shell.

2 Simulating shared memory

In this exercise you are going to build a mini VSM system. VSM stands for Virtual Shared Memory. The idea is the following. Processes created by the system call `fork()` share their text segment, but not their data segment. Therefore, they cannot modify each other's data. However, we can simulate shared memory by the aid of virtual memory, using the call `mprotect()`. As a small demo, you will

make a program with three processes: a parent process, that forks two children. The children should print `ping` and `pong` on standard output in turn. You must implement this by working on a 'shared' variable that is contained in a memory page that is protected using `mprotect`. The parent process should play the role of memory manager, and can send/receive signals from/to the child processes (note that you can use the signals `SIGUSR1` and `SIGUSR2` for any purpose that suits you).
As a start, there are two small programs available on Nestor: `demo.c` and `pingpong.c`. The first demonstrates the use of `mprotect()`, while the latter shows the required functionality using inter process communication using pipes (i.e., without 'shared' memory). Your eventual solution should contain ping/pong processes that look like:

```
void procPingPong(int whoami, volatile int *sharedTurnVariable) {
  /* whoami == 0 or whoami == 1 */
  int msg = 0;
  while (msg < 5) {
    while (*sharedTurnVariable != whoami); /* busy-waiting */
    printf (whoami == 0 ? "Ping\n" : "...Pong\n");
    *sharedTurnVariable = 1 - whoami;
  }
  exit(EXIT_SUCCESS);
}
```

Note that the code contains busy-waiting, which gets extremely inefficient if for every shared access the parent process is accessed. Therefore, you are advised to design a communication protocol which 'locally' caches shared variables.

Apart from the code, you are requested to also briefly describe your design decisions in your report. Especially, pay attention to the communication protocol that you designed (drawing a finite state automation may be a good idea).