UNIVERSITY OF MALTA

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CPS1012: Operating Systems and Systems Programming I
Tutorial Sheet II - Process Control

Author(s): Keith Bugeja
Tutor(s): Adrian De Barro, Mark Magro

Instructions:

1. Make sure you go through your course notes / slides before attempting the exercises.
2. The Unix **man** command is your best friend, **Google search** your second best.
3. Always test function return values for errors; report errors to the standard error stream.

Section A — *This section contains a brief C language refresher and some warm-up exercises.*

Common signatures of the `main` function:

```
int main(int argc, char **argv);
int main(int argc, char **argv, char **env);
```

Preprocessor directive to replace all occurrences of `NAME` with `value`:

```
#define NAME value
```

1. C Refresher

(a) Write a program that outputs `Hello, World!` to the console.

(b) Write a program that uses a looping construct (e.g. `for`) to iterate *n* times, outputting the loop index on each iteration.

(c) Modify the program in 1(b) to encapsulate the loop into a function with the following signature: `void print_string(char *p_string, int p_count, bool p_reverse);` where:

`p_string` is a string to be prefixed to the current iteration number;

`p_count` is the number of iterations;

`p_reverse` determines the order of the loop, be it ascending or descending.

(d) Write a program that compiles to 1(a) if `HELLO_WORLD` is defined or 1(c) otherwise.

Hint: Use `#ifdef`, `#else`, `#endif` and `#define`.

2. Warm-up

(a) Redirect the output of 1(d) to a file called `output.txt`.

(b) Open a new terminal window and redirect the output of 1(d) from the original terminal to the newly opened one.

Hint: Use the `tty` command to find out the name of the new terminal.

(c) Write a program that outputs the current process ID (PID); pipe (|) it into `figlet`.

Note: To install `figlet` type `sudo apt install figlet` in your terminal.

(d) Modify the program in 1(c) to output the parent PID.

(e) Write a program that prints the command line arguments from `argv`.

(f) Write a program that outputs the environment variables from `env`.

(g) Write a program that creates and binds three different exit handlers using `atexit()`. Each handler should output a distinct string.

Section B — *The exercises in this section deal with the creation of processes through* `fork` *and the loading of program binaries into process containers using* `exec`.

1. exec

(a) Write a program that executes the command $\boxed{\texttt{top -d 2 -n 10}}$ using a variant of the `execl` functions.

(b) Modify the program in 1(a) to use an `execv`-type function.

(c) Use either 1(a) or 1(b) to launch a non-existent program; handle the error by displaying the value of $\boxed{\texttt{errno}}$ and outputting the error string using `perror`, and terminate the program with $\boxed{\texttt{EXIT\_FAILURE}}$.

2. `fork`

(a) Write a program that `fork`s and prints the PIDs of both child and parent processes.

(b) Modify the program in 2(a) to `fork` *n* times, where *n*, an integer between 1 and 10, is provided as a command line argument: e.g. $\boxed{\texttt{fork\_multiple <n>}}$. Your program should ascertain that *n* is a valid input; otherwise, report an error and terminate program.

Hint: To convert a string literal to an integer use the $\boxed{\texttt{atoi}}$ (ASCII-to-integer) function.

Consider: This problem can also be solved using recursion.

(c) Write a program that:

i takes an integer argument *t*;
ii `fork`s a child process;
iii delays child process for *t* seconds;
iv `wait`s for child process to terminate;
v outputs the string $\boxed{\text{child terminated}}$ after `wait` returns.

(d) Modify 2(c) to use `waitpid` instead of `wait`; use the `WIFEXITED` and `WEXITSTATUS` macros to acquire additional information about the child process exit status and print the output.

Section C — *In this section, you will familiarise yourselves with the* `fork`*-plus-*`exec` *pattern and develop a small shell to launch arbitrary program binaries.*

Download the linenoise source here:
`https://github.com/antirez/linenoise`

1. `fork`-plus-`exec`

(a) Write a program that `fork`s a child process and `exec`utes $\boxed{\text{ps -f}}$.

(b) Make the parent process in 1(a) `wait` for the child to terminate.

(c) Modify 1(b) to launch the program specified by the command line arguments.

2. `tiny_shell`

(a) Write a program that repeatedly reads user input using the linenoise utility as a `readline` replacement. The program should echo user input.

(b) Modify 2(a) to tokenise the input using whitespace as a separator (ASCII character 32); the program should display the tokenised strings, one token per line.

(c) Use the boilerplate created in 2(b) to create `tiny_shell`, a small command launcher implemented using the `fork`-plus-`exec` pattern.

- i Terminate the shell by entering `exit`.
- ii Do not allow concurrent launching of programs; before launching a new program, the previous must have terminated.
- iii Perform rigorous error checking and handling.
- iv The command prompt should be set to the name of the last program executed followed by $\boxed{\text{ >}}$, e.g. $\boxed{\text{ps >}}$.
