Operating Systems – Exercise 2

Processes, Multiprocessing & IPC

Submission & General Guidelines

- Submission deadline is **25/4/2018, 23:55** Moodle server time
- This exercise must run properly on the provided virtual machine
- Submit your answers in the course website only as single ex2-YOUR_ID.**zip (**e.g. ex2-012345678.zip), containing **only**:
  - ex2.**pdf**
  - ex2.**c**
- Place your name and ID at the top of every source file, as well as in the PDF with the answers.
- No late submission will be accepted!
- Please give concise answers, but make sure to explain all answers.
- Write **clean code** (readable, documented, consistent, ...)

Part 1 (36 points)

In this question we will implement a C application that checks the availability of Internet URLs. The application should receive a file containing a list of URLs, and the number of processes we want to use. After a successful run the application prints a numeric summary of the available URLs, erroneous URLs and unknown (for URLs it failed to test).

1. In addition to the manual pages from exercise 1, read the manual page for the following System Calls:
   a. FORK(2) i.e. execute: `man 2 fork`
   b. PIPE(2) also read PIPE(7)
2. Install the libcurl development package on your VM:

```
sudo apt-get install libcurl3-dev
```

3. Create a new eclipse project for exercise 2, and change the following:
   a. Project->Properties->C/C++ Build->Settings->Tool Settings->Cross GCC Linker->Libraries->Add…->"curl"
4. Complete the application ex2.c (missing parts are marked with //TODO)

Examples (**number of OK/Error/Unknown might vary, even between runs**)

```
$ ./ex2
usage:
        ./ex2 FILENAME NUMBER_OF_PROCESSES

$ ./ex2 top10.txt 1
6 OK, 2 Error, 2 Unknown

$ ./ex2 top10.txt 2
7 OK, 1 Error, 2 Unknown

$ ./ex2 top10.txt 10
8 OK, 2 Error, 0 Unknown
```

Guidelines

- Use the manual
- Make sure to always close the files you are opening
    - Same goes for pipes
- A parent must wait for all his/her child processes
- Always check syscalls return value for errors. ALWAYS.
- You are not allowed to use any additional external libraries, if you are not sure if you are allowed to use something, ask in the forum.
    - You may use any function that was already used in the skeleton
- You may assume that the function input is valid
- **Hint**: a struct is just a bunch of bits, you can write() and read() structs.

Part 2 (24 points)

We will now examine the performance of our program from part 1.

1. Use the provided list of URLs: **top128.txt**
2. Run your program on this file, preceded with the **time(1)** command, using the following number of processes: 1, 2, 4, 8, 16, 32, 64, 128

```
$ time ./ex2 top128.txt 128
87 OK, 15 Error, 26 Unknown

real    0m3.553s
user    0m1.076s
sys     0m0.120s
```

3. Draw a graph (using Google Sheets or Microsoft Excel) with a single series. The graph should show the **real** time each run takes (in milliseconds) [Y axis], as a function of the number of processes [X axis]

4. Explain the graph. Why are we seeing an improvement while only having a single CPU core in our VM? Why, when running a process for each URL the time is significantly larger than 2 seconds (the timeout for each URL check)?
5. Hypothetically, if we change the **check_url** function to only check if the structure of the URL is correct (i.e. http://some.domain/path), will the graph you draw change significantly? why?
    - **IMPORTANT**: Do not make this change. Submission with this change will result in 0 points for this question!

Part 2 (25 points)

What would be the output of the following pseudo C/UNIX code? Explain!

Notes:
Be careful not to explain what the code does! Do not tell a story about x, that it will be multiplied by 2 and then something else will happen.
Give a concise answer about what the output to the screen will be.

A. (12 points)

```
int x; // x is a global variable
int main() {
      int i;
      x = 1;
      for (i = 0; i < 100; i++) {
            x <<= 1;
            fork();
            printf("X=%d\n", x);
      }
      return 0;
}
```

B. (13 points)

```
void sighandler(int);
int main () {

      int pid = getpid();
      signal(SIGINT, sighandler); // Register to signal type Interrupt
      signal(SIGUSR1, sighandler); // Register to signal type User

      printf("PID %d:Pending for signal.\n",pid);
      while(1) { }

   return(0);
}
void sighandler(int signum) {

    // Find out which signal we're handling
    switch (signum) {
        case SIGUSR1:
            printf("Caught User signal from other process \n");
            break;
        case SIGINT:
            printf("Caught Interrupt from  \n");
            exit(0);
            break;
        default:
            fprintf(stderr, "Caught wrong signal: %d\n", signum);
            return;
    }
}
```

Run the above code (you can use eclipse or the terminal for that).
Run the following commands on a different terminal, the PID should be the PID of the process that is created by running the above code:

```
kill -s USR1 PID
kill -s USR1 PID
kill -s INT PID
kill -s INT PID
```

Part 3 (15 points)

For each of the statements below, state whether **it is true or false and explain concisely** (two lines at most):

1. When using fork(), if the parent process exits before the child process then the child process terminates immediately.

2. When using fork(), the child process and the parent process cannot communicate through named pipes. Named pipes are only used for non-related processes.

3. Consider the following pseudo code:

```
int pidfromfork = fork();
int pid1;
If(pidfromfork == 0){
        pid1 = getppid(); //get parent process id
}
else{
        pid1 = getpid(); // get current process id
}
```

   a. pidfromfork and pid1 are equal in the parent process.

   b. pid1 in the parent process and pid1 in the child process are equal.

   c. pidfromfork and pid1 are equal but only in the parent process.