

Operating Systems – Exercise 2

Processes, Multiprocessing & IPC

Submission & General Guidelines

- Submission deadline is **10/4/2019, 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 response time of Internet URLs. The application should receive a file containing a list of URLs and the number of workers we want to use. After a successful run the application prints a numeric average of the successful URL queries, number of sites reached and the number of unknowns (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 (**numbers might vary between runs**)

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

$ ./ex2 top10.txt 1
0.0146 Average response time from 8 sites, 2 Unknown

$ ./ex2 top10.txt 2
0.1002 Average response time from 9 sites, 1 Unknown

$ ./ex2 top10.txt 10
0.0607 Average response time from 9 sites, 1 Unknown
```

Guidelines

- Use the manual.
- Make sure to always close the files you open.
 - 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 provided skeleton.
- You may assume that the function input is valid.
- Feel free to change the internals of given functions, but not their signatures.
- Notice that when compiling using gcc on the terminal, you should still use the “-lcurl” flag.
- **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
0.0949 Average response time from 116 sites, 12 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!

6. How will the results change if we use threads instead of processes (in c/Unix)? In your answer, please refer to overall time, memory use, etc. Explain briefly.

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 = 0; // x is a global variable
int main() {

    while (x <= 100) {
        x++;
        fork();
        printf("x = %d\n", x);
    }

    return 0;
}
```

B. (13 points)

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

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

    if (0 == response) {
        printf("PID1 %d : Pending for signal.\n", getpid());
    } else {
        printf("PID2 %d : Pending for signal.\n", getpid());
    }
    while(1) { }

    return(0);
}

void sighandler(int signum) {

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

What will be the output if we run the given code and, on a different terminal, run the followings commands (replace PID with corresponding numbers):

```
kill -s USR1 PID1
kill -s USR1 PID2
kill -s INT PID1
kill -s USR1 PID1
kill -s INT PID2
```

Briefly explain why. Which part of the output might change?

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 (use Google...).
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 code:

```
public class Worker implements Runnable {  
    @Override  
    public void run() {  
        ...  
    }  
}  
...  
public static void main(String[] args){  
    Thread t = new Thread(new Worker());  
    ??  
}
```

Replacing ?? with `t.run()` and `t.start()` results exactly in the same effects (Google is your best friend).