Operating Systems – Exercise 2

## Processes, Multiprocessing & IPC

### Submission & General Guidelines

* Submission deadline is **10/05/2024, 23:55** Moodle server time.
* 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:
   1. FORK(2) i.e. execute: man 2 fork
   2. PIPE(2) also read PIPE(7)
   3. SIGNAL(7)
2. Install the libcurl development package on your VM:

|  |
| --- |
| sudo apt-get install libcurl3-dev |

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

#### Examples (**numbers might vary between runs**)

|  |
| --- |
| **$ ./ex2**  usage:  ./ex2 NUMBER\_OF\_PROCESSES FILENAME  **$ ./ex2 1 top10.txt**  0.0146 Average response time from 8 sites, 2 Unknown  **$ ./ex2 2 top10.txt**  0.1002 Average response time from 9 sites, 1 Unknown  **$ ./ex2 10 top10.txt**  Illegal url detected, exiting now |

#### 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.
* Do **not** change the program output, i.e. the print statements.
* The check\_url function returns a URL\_ERROR if a URL address doesn’t start with ‘http’. If your program receives a URL\_ERROR, all processes (parent and children) must exit immediately and the following message should be printed: "Illegal url detected, exiting now".
* **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 128 top128.txt**  0.0949 Average response time from 116 sites, 12 Unknown  **real 0m3.553s**  user 0m1.076s  sys 0m0.120s |

1. 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]
2. 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)?
3. 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!

1. 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.

1. **(12 points)**

int x = 0; // x is a global variable

int main() {

while (x <= 100) {

if (fork() != 0){

printf("x = %d\n", x);

}

x++;

}

return 0;

}

1. **(13 points)**

void sighandler(int);

bool continue = false;

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 : Pending for signal.\n”);

} else {

printf(“PID2 : Pending for signal.\n”);

}

while(continue == false) { }

printf("%d : Exiting now.\n", getpid());

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.\n", getpid());

continue = true;

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 INT PID2  
  
Briefly explain why. Which part of the output might change?

### Part 3 (15 points)

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

1. Consider the following code:

int k = 0;

while (k < 10){

k++;

if (fork() == 0){

printf(“A child process created successfully”);

exit(0);

}

}

wait(NULL);

exit(0)

Removing the wait command (wait(NULL);) may cause our program to print less than 10 lines (Assume all fork commands were successful).

1. Two child processes, that created by the same parent using fork() command, cannot communicate through un-named pipe.
2. Inter Process Communication (IPC) is performed faster using Named Pipes compared to using a Memory Mapped File