

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY, VNU HCM
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



OPERATING SYSTEM

Lab 3: Process (Part 1/2)

Pham Minh Tuan - MSSV: 1752595

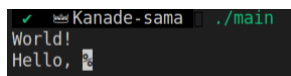
1 Exercise:

1.1 Questions:

1. Describe the return values of `fork()` ?
 - `fork()` returns a negative value if we cannot create a child process.
 - `fork()` returns a zero value if the child process is successfully created.
 - `fork()` returns a positive value, the process ID of the child process, to the parent process.
2. Run the example in Section 4 multiple times and observe the output of each time. Investigate the stability of the outputs and provide reasons as well as solutions for this phenomenon. (Your solution must ensure that the output must always be "Hello, World!")

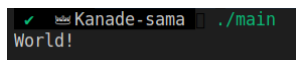
After running the above example multiple times, I receive 3 possible results occurred:

- The first result is shown at below:



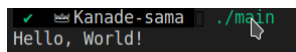
```
✓ Kanade-sama ./main
World!
Hello, 
```

- The second result:



```
✓ Kanade-sama ./main
World!
```

- The third result:



```
✓ Kanade-sama ./main
Hello, World!
```

- The chance for the first and third result appear depends on how the Operating System (OS) first give the control to process parents or child process.
- For the second result, since we the output of the program is shown in terminal which is line-buffered while the program run with full-buffered.
 - When we call `printf()` to print the word into the console, it is not actually that we will immediately print it. The way `printf()` does is called buffering, `printf()` fill the word into the stdout initially.
 - When the buffer is full, or when we print out new line ("`\n`" character), it will be **flushed**. We can force the buffer to flush out immediately by calling `fflush(stdout)`.
 - In our program, after calling `fork()`, we have 2 process running concurrently, in my example the parent process is chosen to run first. However , after running `printf("World!\n")`, the "`\n`" is observed and the output of terminal is line-buffered which means that the buffer will take the words in `printf` until it full or it reaches "`\n`".
 - In addition, we have 2 process running concurrently at the initial time, and every time we run the program in parent or child process especially child process since the `printf()` command does not have "`\n`" character at the end so we do not know which line is stored in the buffer of the process, So sometimes we get the second result.
 - In order to solve the second result, it has better to add "`\n`" at the end of the `printf()` command so it will lead to only the first or third result.
 - To ensure that the output will always be "Hello, World!", I suggest the **wait** command:

```
pid_t child_pid;

child_pid = fork();

//Check if fork did:
if (child_pid == -1){
    //printf("child_pid = %d\n", child_pid);
    perror("Fork");
    exit(1);
}

if (child_pid == 0){
    //fork succeed, we are inside the child process:
    //printf("child_pid = %d\n", child_pid);
    printf("Hello, ");
    fflush(stdout);
}
else
{
    //fork succeed, we are inside the parent process:
    wait(NULL);
    //printf("parents's_pid = %d\n", child_pid);
    printf("World!\n");
    fflush(stdout);
}
```

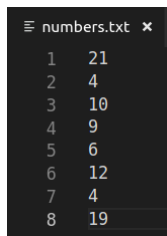
* the *wait(NULL)* command will wait for any child processes complete before running parents process.

3. The result of LINE A is 5 because when we fork the program, there will be 2 processes at the time and the variable *value* will have a independent value in both two processes, in child process it will be 20 and in parent process it will be 5.
4. When a process creates a new process using the `fork()` operation, which of the following states is shared between the parent process and the child process? **Ans:** C. Shared memory segments

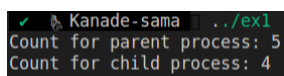
1.2 Programming Exercises

1. Problem 1:

- Numbers.txt



- The result:



2. Problem 2:

- The result:



```
✓ Kanade-sama ./ex2
|_A: pid = 1508
|  |_B: pid = 1509; ppid = 1508
|    |_E: pid = 1510; ppid = 1509
|      |_I: pid = 1511; ppid = 1510
|        |_F: pid = 1512; ppid = 1509
|          |_C: pid = 1513; ppid = 1508
|            |_G: pid = 1514; ppid = 1513
|              |_D: pid = 1515; ppid = 1508
```