Lab 2: Process

Sejong Yoon, Ph.D.

References:

- Silberschatz, et al. *Operating System Concepts* (10e), 2018
- Materials from OS courses offered at TCNJ (Dr. Jikai Li), Princeton, Rutgers, Columbia (Dr. Junfeng Yang), Stanford, MIT, UWisc, VT

Agenda

- Understand fork()
- Understand zombie process
- Understand IPC example using shared memory object

Exercise 1

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int value = 5;
int main()
{
        pid_t pid;
        pid = fork();
        if (pid == 0) { /* child process */
                value += 15;
                return 0;
        }
        else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d", value); /* LINE A */
                return 0;
        }
}
```

← What the output will be?

Exercise 2

```
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
        int i;
        pid_t pid;
        pid = fork();
        if (pid == 0) {
                for (i = 0; i < SIZE; i++) {
                        nums[i] *= -i;
                        printf("CHILD: %d\n",nums[i]);
                }
        }
        else if (pid > 0) {
                wait(NULL);
                for (i = 0; i < SIZE; i++)
                        printf("PARENT: %d\n",nums[i]);
        }
        return 0;
}
```

Exercise 3

- (From textbook, 3.19) Using either a UNIX or a Linux system, write a C program that forks a child process that ultimately becomes a zombie process. This zombie process must remain in the system for at least 10 seconds. Process states can be obtained from the command: ps -l
- The process states are shown below the S column; processes with a state of Z are zombies. The process identifier (pid) of the child process is listed in the PID column, and that of the parent is listed in the PPID column.
- Perhaps the easiest way to determine that the child process is indeed a zombie is to run the program that you have written in the background (using the &) and then run the command ps -l to determine whether the child is a zombie process. Because you do not want too many zombie processes existing in the system, you will need to remove the one that you have created. The easiest way to do that is to terminate the parent process using the kill command. For example, if the process id of the parent is 4884, you would enter: kill -9 4884

Hint

- Remember, zombies are dead (= terminated). However, if parent is also terminated, then the cascading termination is invoked by kernel, and purify zombies along with their parent. So, the key is to let parents alive while children are dead.

Image source: openclipart.org

Exercise 4

- (From textbook 3.21 and 3.22)

The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

$$n = \begin{cases} n/2, & \text{if n is even} \\ 3 \times n + 1, & \text{if n is odd} \end{cases}$$

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if $n = 35$, the sequence is

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C program using the `fork()` system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the `wait()` call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line.

Exercise 4

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>

int main(int argc, char** argv)
{
    int n = atoi(argv[1]);
    pid_t id = fork();
    if (id == 0) {
        while (n > 1) {
            printf("%d ", n);
            if (n % 2 == 0) {
                n = n / 2;
            } else {
                n = 3 * n + 1;
            }
        }
        printf("%d\n", n);
    } else {
        wait(NULL);
    }

    return 0;
}
```

Better to check argument

Child process

Parent process

Exercise 4

In Exercise 3.21, the child process must output the sequence of numbers generated from the algorithm specified by the Collatz conjecture because the parent and child have their own copies of the data. Another approach to designing this program is to establish a shared-memory object between the parent and child processes. This technique allows the child to write the contents of the sequence to the shared-memory object. The parent can then output the sequence when the child completes. Because the memory is shared, any changes the child makes will be reflected in the parent process as well.

This program will be structured using POSIX shared memory as described in Section 3.5.1. The parent process will progress through the following steps:

a. Establish the shared-memory object (`shm_open()`, `ftruncate()`, and `mmap()`).

b. Create the child process and wait for it to terminate.

c. Output the contents of shared memory.

d. Remove the shared-memory object.

Hint

```
int main(int argc, char** argv)
{
    const char* name = "COLLATZ";
    const int SIZE = 4096;
    int n = atoi(argv[1]);
    pid_t id = fork();
    if (id == 0) {
        /* producer */
    } else {
        /* consumer */
    }
    return 0;
}
```

- You need an integer named *shm_fd* to create a shared memory object
- You need a void pointer named *ptr* to the shared memory object data
- You may need an extra header, <sys/mman.h> to use some of the keywords of arguments of mmap()

Lab 2 assignment

- Add a comment to the beginning of your source code containing your name, the name of the course, and the title of the assignment:

  /** John Smith

  CSC345-01 (or CSC345-02)

  Lab 2 Exercise 1 */
- Rename your source file into **lab02_ex1.c, lab02_ex2.c, lab02_ex3.c, lab02_ex4.c**
- Prepare **Makefile** that compiles your source codes into object code **lab02_ex1, lab02_ex2, lab02_ex3, lab02_ex4**
- Zip your source file into **lab02.zip**
- Submit your **zip** file via Canvas