北京郵電大學

实验报告：实验 2 进程控制

——实验内容一

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

2024 年 10 月 14 号

目录

1 实验概述

1.1 实验内容

Collatz 猜想：任意写出一个正整数 N，并且按照以下的规律进行变换：

如果是个奇数，则下一步变成 3N+1；如果是个偶数，则下一步变成 N/2。无论 N 是怎样的一个数字，最终都无法逃脱回到谷底 1。

例如：如果 N=35，则有序列 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1。

实验内容一：

采用系统调用 fork()，编写一个 C 程序，以便在子进程中生成这个序列。

要求：

（1）从命令行提供启动数字

（2）由子进程输出数字序列

（3）父进程等子进程结束后再退出。

1.2 实验环境

1. Windows Subsystem for Linux 2：WSL（Windows Subsystem for Linux）是微软推出的一种在 Windows 操作系统上运行 Linux 的解决方案。WSL 允许用户在 Windows 上运行 Linux 操作系统及其相关的命令行工具和应用程序，而无需使用虚拟机或双重启动配置。
2. Ubuntu 22.04.5 LTS
3. Visual Studio Code 1.94.2：用于连接 wsl 直接进行代码编写，避免使用 vim 等命令行工具，提高编写效率。
4. gcc version 11.4.0

2 程序设计说明

本次实验的目标是编写一个使用 fork() 系统调用的 C 程序，通过子进程生成并输出 Collatz 数列，父进程在子进程执行完毕后退出。

程序设计分为如下几个部分：

2.1 fork()系统调用

fork()用于创建一个新进程。该进程被称为子进程，是父进程的副本，但具有独立的执行流。fork() 的返回值用于区分父进程和子进程。它在父进程中返回子进程的进程 ID，在子进程中返回 0。如果返回负数，则表示创建进程失败。

本实验中使用如下：

```
1. pid_t pid = fork();
```

2.2 用户输入与检查

程序首先通过 scanf() 函数从用户输入中获取一个正整数。如果输入数字小于等于 0，程序会输出错误提示并退出，确保只有正整数被传递给后续的处理逻辑。

具体代码如下：

```
1. int n;
2. printf("Please enter a positive integer: ");
3. scanf("%d", &n);
4.
5. if (n <= 0) {
6.     fprintf(stderr, "Please provide a positive integer greater than
0.\n");
7.     return 1;
8. }
```

2.3 子进程生成 Collatz 数列

在子进程中，调用 generate_collatz_sequence() 函数生成并打印 Collatz 数列。该函数遵循 Collatz 规则：如果当前数字是偶数，则除以 2；如果是奇数，则乘以 3 加 1。函数持续执行，直到数列回到 1 为止。

具体函数实现如下：

```
1. void generate_collatz_sequence(int n) {
2.     while (n != 1) {
3.         printf("%d, ", n);
4.         if (n % 2 == 0) {
5.             n /= 2;
6.         } else {
7.             n = 3 * n + 1;
8.         }
9.     }
10.     printf("1\n");
11. }
```

2.4 父进程等待子进程结束

父进程在调用 fork() 之后，使用 wait() 函数等待子进程完成。当子进程生成并输出完数列后，父进程捕获子进程的结束信号，并在子进程结束后输出提示信息，然后正常退出。

具体代码如下：

```
1. else {
2.     // Parent process
3.     wait(NULL);  // Wait for child to finish
4.     printf("Child process has finished. Parent process exiting.\n");
5. }
```

2.5 错误处理

我设计了两个错误检查：

1. 如果 fork() 失败，程序会输出错误信息并退出，确保进程创建的可靠性。
2. 对用户输入的正整数进行了基本的合法性检查，以防止错误输入导致程序异常运行。

3 程序执行结果

3.1 基本功能测试

输入一个常见的正整数，验证程序能够正确生成 Collatz 数列。

测试用例为 35，结果如下：

```
demo@SevenBill:~$ cd OS/lab2/part1
demo@SevenBill:~/OS/lab2/part1$ gcc -o forktest collatz_fork.c
demo@SevenBill:~/OS/lab2/part1$ ./forktest
Please enter a positive integer: 35
Collatz sequence for 35: 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
Child process has finished. Parent process exiting.
demo@SevenBill:~/OS/lab2/part1$
```

生成序列为 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

并且在子进程输出结束并关闭后，由父进程输出：

```
Child process has finished. Parent process exiting.
```

3.2 边界值测试

输入边界部分的整数（如 1 ， 2），验证程序在边界条件下的表现。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part1$ ./forktest
Please enter a positive integer: 1
Collatz sequence for 1: 1
Child process has finished. Parent process exiting.
demo@SevenBill:~/OS/lab2/part1$ ./forktest
Please enter a positive integer: 2
Collatz sequence for 2: 2, 1
Child process has finished. Parent process exiting.
```

对于测试案例 1，由于输入就是 1，程序应该直接输出 1 而不再进行任何计算。测试通过。

对于测试案例 2，输出为 2，1，通过测试。

且最后都由父进程输出：

```
Child process has finished. Parent process exiting.
```

3.3 异常值测试

验证程序对于非法输入的处理能力。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part1$ ./forktest
Please enter a positive integer: 0
Please provide a positive integer greater than 0.
demo@SevenBill:~/OS/lab2/part1$ ./forktest
Please enter a positive integer: -1
Please provide a positive integer greater than 0.
demo@SevenBill:~/OS/lab2/part1$
```

对于输入 0，验证程序对 0 输入的处理，确保提示输入错误。测试通过。

对于输入-1，验证程序对负数输入的错误提示功能。测试通过。

4 心得总结

本次实验的实验内容一让我对进程创建、系统调用有了更加深入的理解，尤其是 fork() 和 wait() 的使用。整个实验的核心在于通过 fork() 创建子进程，并在子进程中执行 Collatz 数列的生成与输出，父进程则通过 wait() 等待子进程结束。

在一开始，我试图在 Windows 系统里使用 fork()，在报错之后我才意识到本次实验需要 Linux 环境。由于之前使用过 wsl，所以没有选择虚拟机等其他方式，而是直接选择使用 Windows 自带的子系统。经过几次尝试，也成功地配好了 C 语言的环境。

总的来说，本次实验巩固了我对进程控制和系统调用的理解，强化了我在 C 语言编程中的实践能力。在实验过程中，我学会了如何编写更鲁棒的代码、如何进行错误处理以及如何高效地完成子进程与父进程的协作。未来，我会进一步研究进程间通信的更多机制，并将所学应用到更复杂的系统编程任务中。