北京郵電大學

实验报告：实验 2 进程控制

——实验内容二

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。

实验内容二：

以共享内存技术编程实现 Collatz 猜想。

要求在父子进程之间建立一个共享内存对象，允许子进程将序列内容写入共享内存对象，当子进程完成时，父进程输出序列。

父进程包括如下步骤：

建立共享内存对象（shm_open(), ftruncate(), mmap()）

建立子进程并等待他终止

输出共享内存的内容

删除共享内存对象。

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

实验内容二的目的是通过共享内存技术，实现父子进程之间共享 Collatz 数列内容。父进程创建共享内存对象，并允许子进程将生成的 Collatz 数列写入共享内存，父进程读取并输出该内容。

具体设计如下：

2.1 共享内存的创建

在父进程中，通过 shm_open() 创建一个共享内存对象，其名称定义为 /collatz_shm，大小为 4096 字节，权限设置为读写模式。

具体代码如下：

```
1. int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
2. if (shm_fd == -1) {
3.     perror("shm_open");
4.     return 1;
5. }
```

参数解释：

- O_CREAT|O_RDWR 表示如果内存对象不存在，则创建一个新对象，并允许读写操作。
- 0666 表示共享内存对象的权限，即均可读写该内存对象。

接下来通过 ftruncate() 设置共享内存的大小为 4096 字节，以确保共享内存能够存储 Collatz 数列。

具体代码如下：

```
1. if (ftruncate(shm_fd, SHM_SIZE) == -1) {
2.     perror("ftruncate");
3.     return 1;
4. }
```

2.2 共享内存的映射

父进程通过 mmap() 将共享内存映射到自身的地址空间。这样，父子进程都可以通过该映射访问同一个共享内存区域。

具体代码如下：

```
1. char* shared_memory = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE,
                              MAP_SHARED, shm_fd, 0);
2. if (shared_memory == MAP_FAILED) {
3.     perror("mmap");
4.     return 1;
5. }
```

其中，相关参数解释依次如下：

- 0 表示内存起始地址由操作系统决定。
- SHM_SIZE 指定映射的大小，即 4096 字节。
- PROT_READ | PROT_WRITE 设置映射区域为可读可写。
- MAP_SHARED 表示该映射为共享映射，父子进程都可以访问这片内存区域。
- shm_fd 是共享内存对象的文件描述符，指示 mmap() 将该共享内存映射到进程空间中。
- 偏移量为 0，表示从共享内存对象的开头开始映射。

2.3 子进程生成 Collatz 数列

在创建子进程后，子进程调用 generate_collatz_sequence() 函数生成数列，并将结果写入共享内存。

函数具体实现如下：

```
1. void generate_collatz_sequence(int n, char* shared_memory) {
2.     char buffer[256];  // 用于存储每个数字转换后的字符串
3.     while (n != 1) {
4.         sprintf(buffer, "%d, ", n);
5.         strcat(shared_memory, buffer);
6.         if (n % 2 == 0) {
7.             n /= 2;
8.         } else {
9.             n = 3 * n + 1;
10.        }
11.    }
12.    strcat(shared_memory, "1\n");
13. }
```

其中，Collatz 的猜想部分已在实验内容一中讲解，故不做赘述。

sprintf() 用于将数字转换为字符串形式。strcat() 将生成的字符串逐步拼接到共享内存中，以便父进程稍后读取。

2.4 父进程输出共享内存内容

父进程在等待子进程结束后，从共享内存中读取数列并输出。代码如下：

```
1. wait(NULL);
2. printf("Collatz sequence for %d: %s", n, shared_memory);
```

2.5 清理共享内存

最后，父进程使用 shm_unlink() 删除共享内存对象，防止系统资源泄漏。

```
1. if (shm_unlink(SHM_NAME) == -1) {
2.     perror("shm_unlink");
3.     return 1;
4. }
```

3 程序执行结果

为了验证程序的正确性，分别对不同类型的输入进行测试。

3.1 基本功能测试

输入一个常见的正整数，验证程序是否能生成正确的 Collatz 数列。

测试用例为 35 和 114514，结果如下：

```
demo@SevenBill:~/OS/lab2/part2$ ./sharetest.exe
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/part2$ ./sharetest.exe
Please enter a positive integer: 114514
Collatz sequence for 114514: 114514, 57257, 171772, 85886, 42943, 128830, 64415, 193246, 96623, 2
89870, 144935, 434806, 217403, 652210, 326105, 978316, 489158, 244579, 733738, 366869, 1100608, 5
50304, 275152, 137576, 68788, 34394, 17197, 51592, 25796, 12898, 6449, 19348, 9674, 4837, 14512,
7256, 3628, 1814, 907, 2722, 1361, 4084, 2042, 1021, 3064, 1532, 766, 383, 1150, 575, 1726, 863,
2590, 1295, 3886, 1943, 5830, 2915, 8746, 4373, 13120, 6560, 3280, 1640, 820, 410, 205, 616, 308,
 154, 77, 232, 116, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
Child process has finished. Parent process exiting.
demo@SevenBill:~/OS/lab2/part2$
```

子进程生成序列皆正确。并且由代码可知，这些输出均为父进程的输出。

3.2 边界值测试

输入边界部分的整数，验证程序在边界条件下的表现。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part2$ ./sharetest.exe
Please enter a positive integer: 1
Collatz sequence for 1: 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。

且序列由子进程生成，写入共享内存，再由父进程输出，测试通过。

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/part2$ ./sharetest.exe
Please enter a positive integer: -5
Please provide a positive integer greater than 0.
demo@SevenBill:~/OS/lab2/part2$ ./sharetest.exe
Please enter a positive integer: 0
Please provide a positive integer greater than 0.
```

对于输入-5，验证程序对负数输入的错误提示功能。测试通过。

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。

4 心得总结

通过本次实验，我深入学习了共享内存技术及其在父子进程之间的通信应用。

通过 shm_open()、ftruncate() 和 mmap()，我理解了如何在多个进程间共享数据。这种技术在进程间通信中非常高效，尤其是在需要频繁数据传输的场景下。这种技术是我以前没有接触过的，也让我认识到自己知识的不足。

同实验内容一，父进程通过 wait() 等待子进程结束，确保了父子进程间的任务协调和数据的有序输出。这也使我对进程同步的概念有了更清晰的理解。

本实验强调了共享内存资源的正确创建和清理，避免系统资源泄漏。在实际开发中，良好的资源管理是确保程序稳定运行的关键。

通过这些系统调用的结合使用，我掌握了进程间通信的基本技巧，为后续更复杂的系统编程打下了坚实基础。