操作系统实验二：进程控制
实验报告

毛子恒
2019211397
北京邮电大学 计算机学院

日期：2021 年 10 月 17 日

1 概览

1.1 实验内容

1.1.1 实验内容一

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，最终都会变成 1。

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。

1.1.2 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：

1. 建立共享内存对象（`shm_open()`,`ftruncate()`,`mmap()`）
2. 建立子进程并等待他终止。
3. 输出共享内存的内容。
4. 删除共享内存对象。

1.1.3 实验内容三

设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一个进程。比如，第一个进程发

消息："I am Here"，第二个进程收到后，将它改变为："i AM hERE"之后，再发给第一个进程。

1.2 实验环境

- openEuler 20.03 64bit with ARM
- gcc version 7.3.0
- vim 8.1

2 实验设计

2.1 实验内容一

2.1.1 相关 API

- `pid_t fork(void);`：创建子进程，子进程是调用该函数的进程的拷贝。子进程与父进程有各自单独的地址空间，在调用之后其中的内容是相同的。如果创建成功，该函数返回给父进程以子进程的 PID，给子进程返回 0。
- `pid_t wait(int *wstatus);`：暂停调用该函数的进程直到所有子进程都终止，当成功时，返回终止的子进程的 PID，失败时返回-1。

2.1.2 设计概述

如图 1所示，父进程首先通过标准输入读取 n，之后通过 `fork()` 函数创建子进程，并调用 `wait()` 函数等待子进程结束。

子进程得到与父进程数据段地址空间的一份副本，其中包含 n 的值。之后子进程通过循环计算序列并且输出，之后通过`return 0`终止。

父进程回收子进程后，输出一行字符串之后终止。

2.2 实验内容二

2.2.1 相关 API

- `int shm_open(const char *name, int oflag, mode_t mode);`:打开(或者创建新的同时打开）一个 POSIX 共享内存对象。共享内存对象可以作为一个句柄提供给另一个进程来访问一块共享内存。`text` 参数用于指定一个共享内存对象的名字，推荐格式为`/somename`；`oflag` 用于指定一些选项，例如 `O_RDONLY` 表示对象只读，`O_RDWR` 表示对象可读可写，`O_CREAT` 表示如果对象不存在则创建一个新对象，新对象的权限通过 `mode` 参数设置。如果成功，该函数返回一个文件描述符（一个非负整数）。
- `int ftruncate(int fd, off_t length);`：通过该函数给文件（即之前创建的共享内存对象）分配空间，`fd` 表示文件描述符，`length` 为裁切的长度。如果成功，返回 0.
- `void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);`：在当前进程的虚拟地址空间中创建一个映射，`addr` 指明映射的起始地址，若为 `NULL`，则内

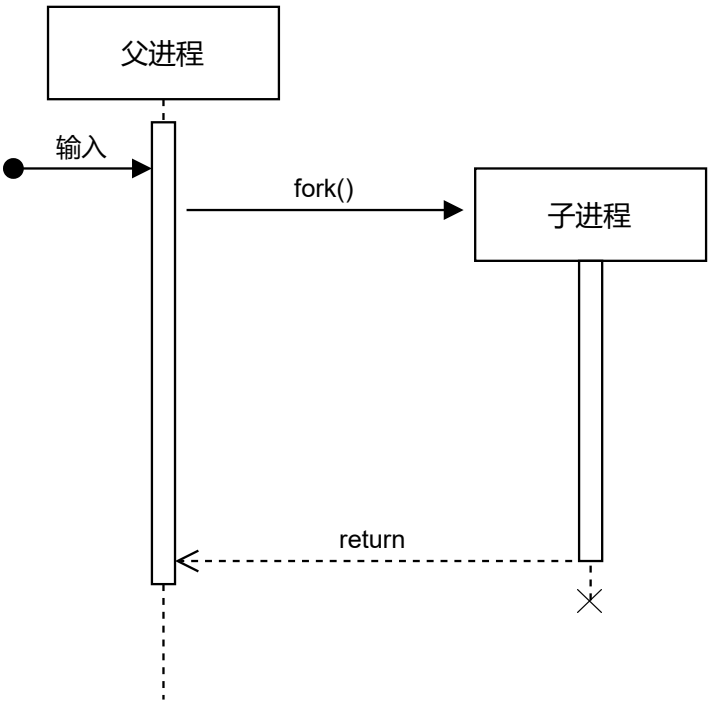

图 1: 实验内容一时序图

核自动选择一个页对齐的地址创建映射；`length` 指明映射的长度；`prot` 指明对映射的内存保护选项，包括 `PROT_EXEC` 可执行、`PROT_READ` 可读、`PROT_WRITE` 可写、`PROT_NONE` 不可访问；`flags` 决定对映射的修改是否对其他进程可见，并且修改是否会被代入到构成映射的文件，`MAP_SHARED` 表示对该映射的修改会分享给其他进程；`fd` 为文件描述符；`offset` 为文件中起始位置的偏移量。当映射成功时，该函数返回一个指向映射区域的指针。

- `int munmap(void *addr, size_t length);`：删除给定地址的映射，之后对于给定地址的引用会无效。当进程终止时，映射也会被自动删除。
- `int shm_unlink(const char *name);`：移除一个共享内存对象名称，如果所有的进程都已经删除了对该对象的映射，那么该内存区域会被释放。

2.2.2 设计概述

流程仍然如**图 1**所示，父进程首先读取 n，创建共享内存对象并且分配空间，创建对共享内存的映射，之后通过 `fork()` 函数创建子进程，并调用 `wait()` 函数等待子进程结束。

子进程得到了 n 和共享内存的地址，计算循环计算序列并且以字符串的形式打印到共享内存中，之后删除映射并通过`return 0`终止。

父进程回收子进程后，向标准输出打印共享内存中的字符串，之后删除映射并且移除共享内存对象（此时会释放共享内存），最后终止。

2.3 实验内容三

2.3.1 相关 API

- `int pipe(int pipefd[2]);`：创建一个单向的管道，`pipefd` 用于返回管道两端的文件描

述符，分别为读取端和写入端。如果成功，函数返回 0。

- `int close(int fd);`：关闭一个文件描述符。
- `ssize_t write(int fd, const void *buf, size_t count);`：向文件描述符 `fd` 指明的文件写入从 `buf` 开始的 `count` 个字节。若成功，返回写入的字节数。
- `ssize_t read(int fd, void *buf, size_t count);`：从文件描述符 `fd` 指明的文件读取到从 `buf` 开始的缓冲区，最多读取 `count` 个字节。若成功，返回读取的字节数。

2.3.2 设计概述

如图 2所示，父进程首先创建两个管道，之后通过 `fork()` 函数创建子进程。

子进程得到与父进程数据段地址空间的一份副本，包含两个管道的入口和出口描述符。子进程首先关闭上行（子进程到父进程）管道的读取端和下行（父进程到子进程）管道的写入端，之后子进程调用 `read()` 函数阻塞，等待下行管道的输入。

父进程首先关闭上行管道的写入端和下行管道的读取端，之后从标准输入中读入一行字符串，并且写入到下行管道，之后调用 `read()` 函数阻塞，等待上行管道的输入。

子进程得到下行管道的输入后，对字符串进行大小写转换，之后写入到上行管道，并且关闭剩余的文件描述符并终止。

父进程得到上行管道的输入后，向标准输出打印字符串，关闭剩余的文件描述符并终止。

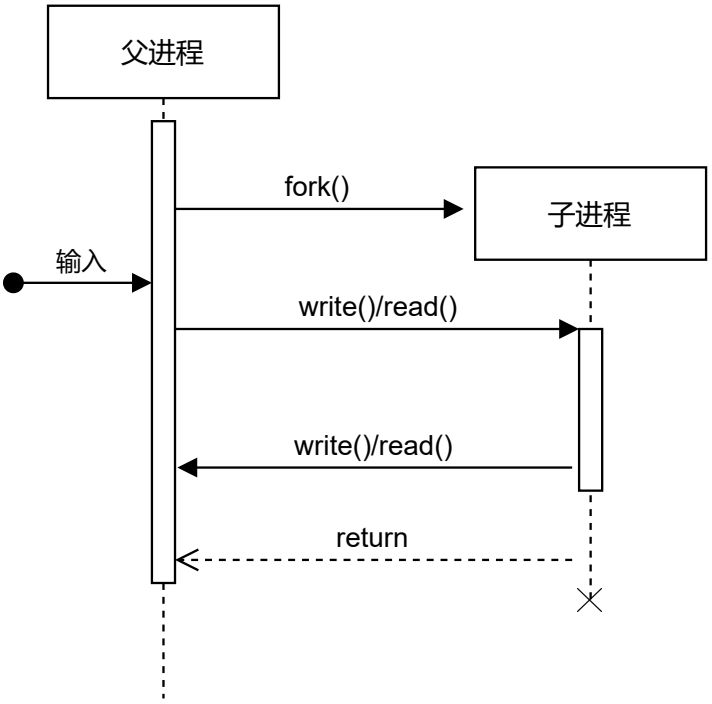

图 2: 实验内容三时序图

3 运行结果及分析

3.1 实验内容一

3.1.1 输入

```
35
```

3.1.2 输出

```
35 106 53 160 80 40 20 10 5 16 8 4 2 1
Done.
```

3.1.3 分析

主进程成功创建了子进程，子进程计算并输出了序列。

3.2 实验内容二

输入和输出与实验内容一相同，略。

3.2.1 分析

主进程成功创建了共享内存和子进程，子进程计算了序列并且输出到共享内存，主进程打印了共享内存的内容。

3.3 实验内容三

3.3.1 输入

```
I am Here
```

3.3.2 输出

```
i AM hERE
```

3.3.3 分析

主进程和子进程之间成功用管道交换信息。

4 实验总结

本次实验中我利用 POSIX API 编写了三个与进程控制和通信有关的程序，使我对相关知识点的掌握更加牢固。

我通过查阅Linux man page获取相关 API 的信息，并且参考教材的内容编写程序，编写过程中除了头文件和链接选项没有遇到太多问题。在阅读 man page 和相关文档的过程中我对进程控制和共享内存模型的有了更多的认识。

本次实验使我的 C 编程能力和英文文献阅读能力得到提高，我从中收获颇丰。