

深圳大学实验报告

课程名称： 操作系统

实验项目名称： 处理机调度

学院： 计算机与软件学院

专业： 软件工程

指导教师： 张 滇

报告人： 郑彦薇 学号： 2020151022 班级： 软件工程 01 班

实验时间： 2023 年 04 月 13 日

实验报告提交时间： 2023/4/14

教务处制

一、实验目的与要求：

1. 掌握进程同步与通信实验
2. 加深对进程同步与通信操作的直观认识
3. 掌握 Linux 操作系统的进程、线程机制和编程接口
4. 掌握 Linux 操作系统的进程和线程间的同步和通信机制
5. 掌握经典同步问题的编程方法

二、方法、步骤：

1. 相关知识内容：进程、线程间的同步和通信机制，同步问题

进程同步是操作系统提供的一种同步机制，用于协调多个并发执行进程的工作先后次序。线程同步是指在多线程环境中，控制多个线程之间的执行顺序，保证程序的正确性和稳定性。

进程同步和通信机制：

- ①信号：进程可以向另一个进程发送信号，用于通知某个事件的发生。
- ②消息队列：一种进程间的通信机制，可以实现不同进程之间的数据传递。
- ③信号量：一种进程间同步的机制，可以用来保护共享资源的访问。
- ④共享内存：一种高效的进程间通信机制，多个进程可以共享同一块物理内存。

线程同步和通信机制：

- ①互斥锁：一种线程同步机制，可以保证在同一时间只有一个线程可以访问共享资源。
- ②读写锁：可以实现读操作的并发访问和写操作的独占访问。
- ③信号量：实现多个线程之间对共享资源的互斥访问。

同步问题：指在多线程环境下，为避免竞争条件的出现而设计的一些典型问题，包括生产者消费者问题、哲学家就餐问题、读写者问题等。这些问题的本质都是在多个线程之间共享资源的情况下，如何实现线程之间的协调和同步，避免相互之间的干扰和竞争，保证程序的正确性和稳定性。

2. 程序设计的思路和方法：

根据实验所给的代码框架，按照以下思路对代码进行补充：

在生产者 `producer` 中，设置一个循环，不断读入客户端输入的信息，将其保存到一个缓冲区并等待消费者 `customer` 中的读取进程将数据读走。在 `customer` 中，通过 `fork` 创建两个进程，同样设置一个循环，在循环中两个进程并发交替执行，读取缓冲区的信息并在读取后将缓冲区清空，两个进程打印相应的信息以观察交替执行的过程。

退出程序的方法：判断当前缓冲区中的信息是否是“quit”，如果是，则退出当前循环，并释放信号量。在生产者程序中，需要结束共享内存的挂载映像，删除共享内存区域。

对于缓冲区已满而生产者继续生产数据或缓冲区为空时消费者继续消费数据的情况，使用信号量来控制生产者线程和消费者线程的数量，当缓冲区已满时，生产者线程需要等待信号量的释放；当缓冲区为空时，消费者线程需要等待信号量的释放。

三、实验过程及内容：

根据上述程序设计思路和方法，对实验所给的代码框架进行补充。

1. 头文件 shm_com_sem.h 程序代码及代码说明如下：

```
1 #include<fcntl.h>
2 #include<sys/stat.h>
3 #include<semaphore.h>
4
5 #define LINE_SIZE 256
6 #define NUM_LINE 16
7
8 //用于创建信号量时的识别id
9 const char * queue_mutex = "queue_mutex";
10 const char * queue_empty = "queue_empty";
11 const char * queue_full = "queue_full";
12
13 //生产者消费者公用的缓冲区，含读写指针line_write和line_read，以及缓冲数据区buffer[NUM_LINE][LINE_SIZE]，buffer[X]指向第X行信息，
    buffer[X][Y]指向X行信息的第Y个字符
14
15 struct shared_mem_st{
16     char buffer[NUM_LINE][LINE_SIZE];
17     int line_write;
18     int line_read;
19 };
```

在本段代码中，共享内存结构体 shared_mem_st 用于存储生产者和消费者之间共享的数据，包括缓冲数据区和读写指针。

所定义的 3 个用于创建信号量时的识别 id，分别用于保护缓冲区的互斥访问、空闲等待和缓冲区已满的等待。

2. Producer.c 程序代码及代码说明如下：

头文件：

```
1 #include<unistd.h>
2 #include<stdlib.h>
3 #include<stdio.h>
4 #include<string.h>
5 #include<sys/shm.h>
6 #include "shm_com_sem.h"
7
```

主函数中：

```
10 //共享内存（缓冲区指针）
11 void *shared_memory = (void *)0;
12 struct shared_mem_st *shared_stuff;
13 //将无类型共享存储区转换为shared_mem_st类型的指针
14 char key_line[256];
15 //共享内存的id
16 int shmid;
17 //访问共享内存的互斥量、空缓冲区、满缓冲区信号量。皆为信号量指针
18 sem_t *sem_queue, *sem_queue_empty, *sem_queue_full;
19
20 //遮住部分：获取共享内存区，并挂入内存
21 //shmget函数：得到一个共享内存标识符或创建一个共享内存。IPC_CREAT：创建新的共享内存
22 shmid = shmget((key_t)1234, sizeof(struct shared_mem_st), 0666| IPC_CREAT);
23
24 if(shmid == -1){
25     fprintf(stderr, "shmget failed\n");
26     exit(EXIT_FAILURE);
27 }
28
29 //shmat函数：挂接共享内存，将共享内存段连接到进程地址空间
30 shared_memory = shmat(shmid, (void*)0, 0);
31 //得到一个缓冲区指针shared_memory
32 if(shared_memory == (void *)-1){
33     fprintf(stderr, "shmat failed\n");
34     exit(EXIT_FAILURE);
35 }
36
37 //将缓冲区指针转换为shared_mem_st类型的指针shared_stuff
38 shared_stuff = (struct shared_mem_st*)shared_memory;
39
```

```

40 //遮住部分：创建三个信号量
41 //sem_queue：保护缓冲区的互斥访问
42 sem_queue = sem_open(queue_mutex, O_CREAT, 0666, 1);
43 //sem_queue_empty：等待空缓冲区
44 sem_queue_empty = sem_open(queue_empty, O_CREAT, 0666, 16);
45 //sem_queue_full：等待缓冲区已满
46 sem_queue_full = sem_open(queue_full, O_CREAT, 0666, 0);
47
48 //读写指针初始化，开始时都指向第0行
49 shared_stuff->line_write = 0;
50 shared_stuff->line_read = 0;
51
52 //不断从控制台读入按键输入的字符行
53 while(1){
54     //提示可以输入，并用gets()读入按键行到key_line中
55     printf("Enter your text('quit' for exit): ");
56     fgets(key_line, 256, stdin);
57
58     //遮住部分：将输入的行写入缓冲区，要有信号量操作
59     sem_wait(sem_queue_empty); //信号量>0则-1，表示占用一个空缓冲区，并立即返回
60     strcpy(shared_stuff->buffer[shared_stuff->line_write], key_line);
61
62     //改变写指针shared_stuff->line_write的值
63     shared_stuff->line_write = (shared_stuff->line_write+1) % NUM_LINE;
64     //sem_post释放sem_queue_full信号量，将其值+1，表示有一个缓冲区已被写满
65     sem_post(sem_queue_full);
66
67     //如果键入"quit"则退出
68     if(strcmp(key_line, "quit\n") == 0){
69         break;
70     }
71 }
72
73 //遮住部分
74 //释放信号量，sem_unlink函数从系统中删除有名信号量
75 sem_unlink(queue_mutex);
76 sem_unlink(queue_empty);
77 sem_unlink(queue_full);
78
79 //结束共享内存存在本进程的挂载映像
80 if(shmdt(shared_memory) == -1){
81     fprintf(stderr, "shmdt failed\n");
82     exit(EXIT_FAILURE);
83 }
84 //删除共享内存区域
85 if(shmctl(shmid, IPC_RMID, 0) == -1){
86     fprintf(stderr, "shmctl(IPC_RMID) failed\n");
87     exit(EXIT_FAILURE);
88 }
89 exit(EXIT_SUCCESS);

```

这段代码实现了一个生产者进程，负责将用户在控制台输入的字符行写入共享内存区，以供其他消费者进程读取。

3. Customer.c 程序代码及代码说明如下：

头文件：

```

1 #include<unistd.h>
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<string.h>
5 #include<sys/shm.h>
6 #include<sys/wait.h>
7 #include "shm_com_sem.h"
8 #include<signal.h>
9

```

主函数中：

```

12 void *shared_memory = (void *)0;
13 struct shared_mem_st *shared_stuff;
14
15 int shmid;
16 int num_read;
17 pid_t fork_result;
18 sem_t *sem_queue, *sem_queue_empty, *sem_queue_full;
19

```

```

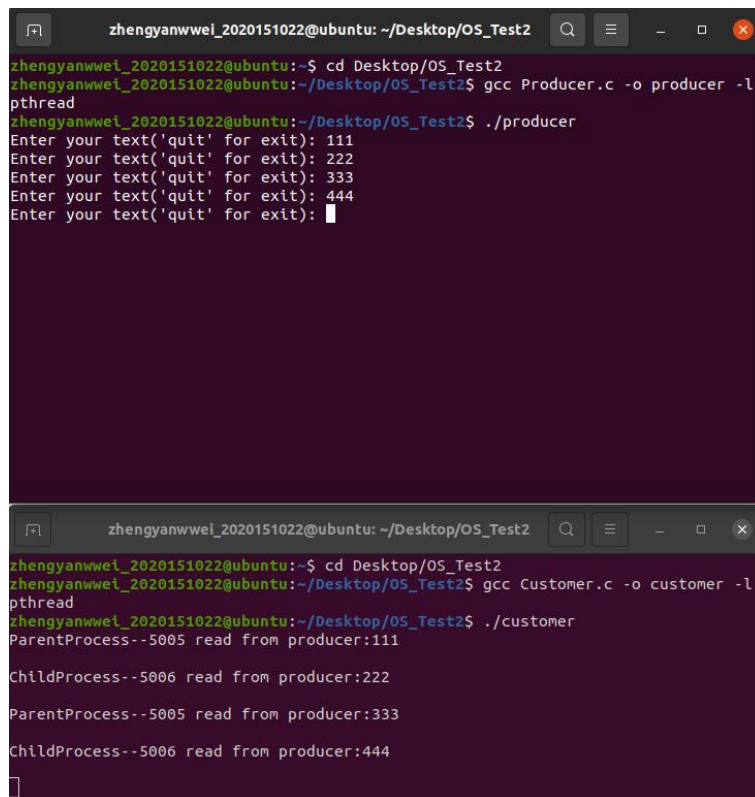
20 //遮住部分：获取共享内存区，并挂入内存
21 shmid = shmget((key_t)1234, sizeof(struct shared_mem_st), 0666| IPC_CREAT);
22 if(shmid == -1){
23     fprintf(stderr, "shmget failed\n");
24     exit(EXIT_FAILURE);
25 }
26
27 shared_memory = shmat(shmid, (void *) 0, 0);
28
29 if(shared_memory == (void*) -1){
30     fprintf(stderr, "shmat failed\n");
31     exit(EXIT_FAILURE);
32 }
33
34 //将缓冲区指针转化为shared_mem_st类型
35 shared_stuff = (struct shared_mem_st *) shared_memory;
36
37 //获取producer创建的3个信号量，根据名字(queue_mutex、queue_empty、queue_full)识别。通过生产者进程和消费者进程对这3个信号量的
    操作，实现对缓冲区的同步和互斥
38 //queue_mutex：用于互斥访问缓冲区，同一时间只能有一个进程访问缓冲区
39 sem_queue = sem_open(queue_mutex, 0_CREAT);
40 //queue_empty：用于表示缓冲区中空的空间数，即生产者进程每放入一个数据，就会触发该信号量；消费者进程每读取一个数据，就会释放该信号量
41 sem_queue_empty = sem_open(queue_empty, 0_CREAT);
42 //queue_full：用于表示缓冲区中已经有数据的空间数，即消费者进程每读取一个数据，就会触发该信号量；生产者进程每放入一个数据，就会释放
    该信号量。
43 sem_queue_full = sem_open(queue_full, 0_CREAT);
44
45 //创建了两个进程
46 fork_result = fork();
47 if(fork_result == -1){
48     fprintf(stderr, "Fork failure\n");
49 }
50 if(fork_result == 0){ //child
51     while(1){
52         //遮住部分：信号量操作，打印消费内容及进程号
53         sem_wait(sem_queue_full);
54         sem_wait(sem_queue);
55
56         printf("ChildProcess--%d read from producer:%s\n", getpid(), shared_stuff->buffer[shared_stuff-
            >line_read]);
57
58         //发现quit退出
59         if(strcmp(shared_stuff->buffer[shared_stuff->line_read], "quit\n") == 0)
60             break;
61
62         shared_stuff->line_read =(shared_stuff->line_read+1) % NUM_LINE;
63         sem_post(sem_queue);
64         sem_post(sem_queue_empty);
65     }
66     //遮住部分：释放信号量
67     sem_unlink(queue_mutex);
68     sem_unlink(queue_empty);
69     sem_unlink(queue_full);
70
71 }else{ //parent，与子进程相似
72     while(1){
73         sem_wait(sem_queue_full);
74         sem_wait(sem_queue);
75
76         printf("ParentProcess--%d read from producer:%s\n", getpid(), shared_stuff->buffer[shared_stuff-
            >line_read]);
77
78         if(strcmp(shared_stuff->buffer[shared_stuff->line_read], "quit\n") == 0){
79             break;
80         }
81
82         shared_stuff->line_read =(shared_stuff->line_read+1) % NUM_LINE;
83         sem_post(sem_queue);
84         sem_post(sem_queue_empty);
85     }
86     sem_unlink(queue_mutex);
87     sem_unlink(queue_empty);
88     sem_unlink(queue_full);
89 }
90 exit(EXIT_SUCCESS);

```

这段代码实现了一个消费者进程，从共享内存区中的缓冲区中读取数据并输出到终端，通过信号量进行同步与互斥，与生产者进程共同维护缓冲区。

四、实验结论：

1. 在两个终端中分别运行 Producer.c 的编译文件和 Customer.c 的编译文件，然后在生产者进程中输入相应的信息，可以看到消费者的两个进程交替执行，打印缓冲区的数据



The first terminal window shows the compilation and execution of the Producer program. The user enters 'quit' to exit, and the program prints the values 111, 222, 333, and 444. The second terminal window shows the compilation and execution of the Customer program. The parent process reads the values 111, 333, and 444, while the child process reads 222 and 444.

```
zhengyanwei_2020151022@ubuntu: ~/Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~$ cd Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ gcc Producer.c -o producer -l pthread
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./producer
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): 333
Enter your text('quit' for exit): 444
Enter your text('quit' for exit):

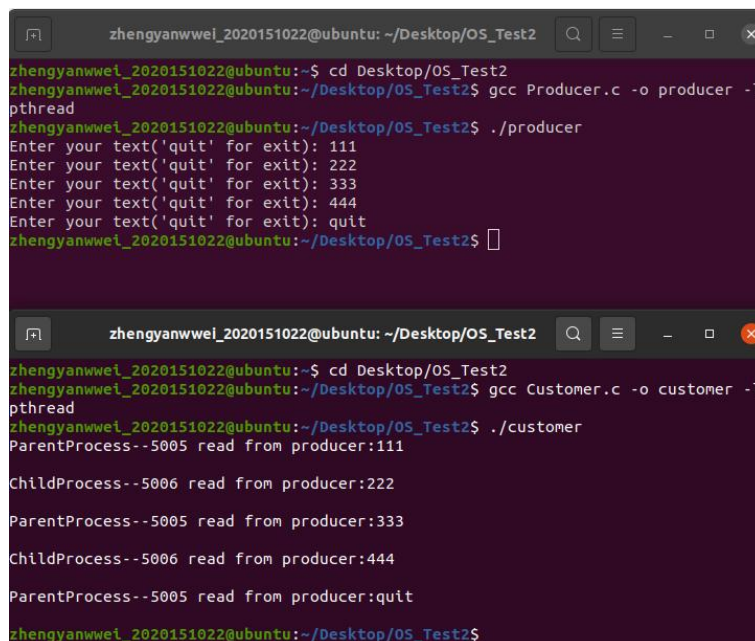
zhengyanwei_2020151022@ubuntu: ~/Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~$ cd Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ gcc Customer.c -o customer -l pthread
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./customer
ParentProcess--5005 read from producer:111

ChildProcess--5006 read from producer:222

ParentProcess--5005 read from producer:333

ChildProcess--5006 read from producer:444
```

2. 父进程接收 quit 退出：



The first terminal window shows the compilation and execution of the Producer program. The user enters 'quit' to exit, and the program prints the values 111, 222, 333, and 444. The second terminal window shows the compilation and execution of the Customer program. The parent process reads the values 111, 333, and 444, while the child process reads 222 and 444. The parent process also reads 'quit' and prints it.

```
zhengyanwei_2020151022@ubuntu: ~/Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~$ cd Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ gcc Producer.c -o producer -l pthread
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./producer
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): 333
Enter your text('quit' for exit): 444
Enter your text('quit' for exit): quit
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$

zhengyanwei_2020151022@ubuntu: ~/Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~$ cd Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ gcc Customer.c -o customer -l pthread
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./customer
ParentProcess--5005 read from producer:111

ChildProcess--5006 read from producer:222

ParentProcess--5005 read from producer:333

ChildProcess--5006 read from producer:444

ParentProcess--5005 read from producer:quit

zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$
```

3. 子进程接收 quit 退出：

对于上述代码，当我想在子进程接收 quit 时实现退出时，会发现此时只有生产者能够正常退出。


```
zhengyanwei_2020151022@ubuntu:~/Desktop/05_Test2$ ./producer
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): quit
zhengyanwei_2020151022@ubuntu:~/Desktop/05_Test2$

zhengyanwei_2020151022@ubuntu:~/Desktop/05_Test2$ ./customer
ParentProcess--5012 read from producer:111

ChildProcess--5013 read from producer:222

ParentProcess--5012 read from producer:111

ChildProcess--5013 read from producer:quit
```

父进程等待

这是因为在我的消费者进程中，子进程和父进程的信息打印、释放信号量等操作相对独立，当前进程为子进程时，子进程退出后，父进程会继续等待信号量并读取共享内存中的数据，直到接收到 quit，但此时生产者进程已经退出。

为了实现无论是子进程接收 quit 还是父进程接收 quit，都能退出消费者进程，即生产者输入 quit，消费者中的所有进程都退出，我使用 kill 函数向所有进程发送 SIGKILL 信号，强制结束所有进程。

```
87     kill(0, SIGKILL);
88     exit(EXIT_SUCCESS);
```

此时再次分别尝试在子进程和父进程接收 quit 时退出程序，可以看到在使用 kill 函数的情况下，程序正常退出。

```
zhengyanwei_2020151022@ubuntu: ~/Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~$ cd Desktop/OS_Test2
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./producer
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): quit
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./producer
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): 111
Enter your text('quit' for exit): 222
Enter your text('quit' for exit): quit
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$

pthread
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./customer
ParentProcess--2179 read from producer:111

ChildProcess--2180 read from producer:222

ParentProcess--2179 read from producer:111

ChildProcess--2180 read from producer:quit
Killed
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$ ./customer
ParentProcess--2201 read from producer:111

ChildProcess--2202 read from producer:222

ParentProcess--2201 read from producer:111

ChildProcess--2202 read from producer:222

ParentProcess--2201 read from producer:quit
Killed
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test2$
```

五、实验体会：（根据自己情况填写）

1. 通过本次实验，对进程、线程之间的同步和通信机制有了一定的掌握。
2. 学会如何应用信号量、设置读写指针等解决同步问题中会出现的竞争条件。
3. 通过对进程代码框架的补充，掌握共享内存的创建方法以及如何将其映射到本进程的进程空间、如何释放信号量。

指导教师批阅意见：

成绩评定：

指导教师签字：

年 月 日

备注：