



中山大學

SUN YAT-SEN UNIVERSITY

实 验 报 告

课程名称: 操作系统

姓 名: 孙广岩

学 号: 20354242

专业班级: 智能科学与技术专业 5 班

任课教师: 吴贺俊

2022 年 11 月 4 日

实验报告成绩评定表

评定项目	内 容	满 分	评 分	总 分
实验态度	态度端正、遵守纪律、出勤情况	10		
实验过程	按要求完成算法设计、代码书写、 注释清晰、运行结果正确	30		
实验记录	展示讲解清楚、任务解决良好、实 验结果准确	20		
报告撰写	报告书写规范、内容条理清楚、表 达准确规范、上交及时、无抄袭， 抄袭记 0 分，提供报告供抄袭者扣 分。	40		
评语：				
指导老师签字： 年 月 日				

实验 4.1 互斥控制

一、实验目的

1. 进一步认识并发执行的实质，学习解决进程互斥的方法。

二、实验内容

1. 任务描述

- 用进程或线程实现前面的两个算法，算法 A 和 B，其中：
 - ♦ 算法 A：算法一到四任选一个；
 - ♦ 算法 B：Dekker 算法或者 Peterson 算法
- 观察算法多次运行，检测是否有同时进入临界区的情况，分析原因，撰写报告。

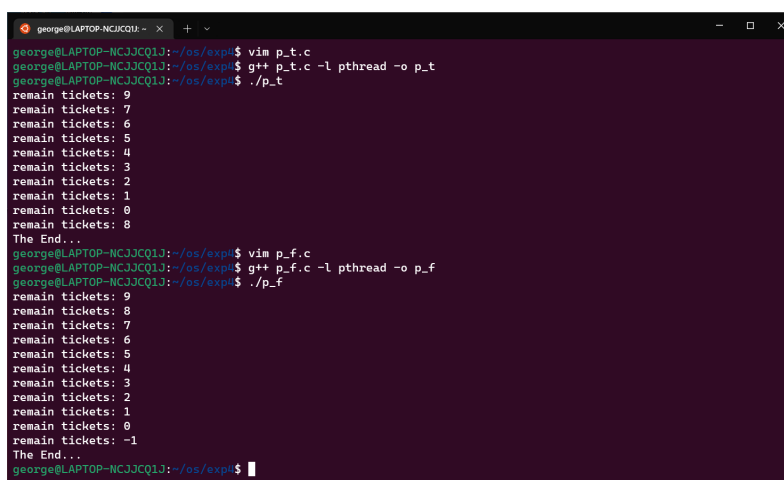
2. 实验方案

我实现了不使用算法正确，不使用算法错误，算法二正确，算法二错误和 Peterson 算法，分别命名为 p_t.c, p_f.c, p2_t.c, p2_f.c, peterson.cpp。

我稍微修改了一下实验的条件，我是设置的与 [url](#) 的条件是一样的，这样的设置可以更加简单的看出线程是否同时进入了临界区。

三、实验记录

1. 无算法



```
george@LAPTOP-NCJJCQ1J: ~$ vim p_t.c
george@LAPTOP-NCJJCQ1J: ~$ gcc p_t.c -l pthread -o p_t
george@LAPTOP-NCJJCQ1J: ~$ ./p_t
remain tickets: 9
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
remain tickets: 8
The End...
george@LAPTOP-NCJJCQ1J: ~$ vim p_f.c
george@LAPTOP-NCJJCQ1J: ~$ gcc p_f.c -l pthread -o p_f
george@LAPTOP-NCJJCQ1J: ~$ ./p_f
remain tickets: 9
remain tickets: 8
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
remain tickets: -1
The End...
george@LAPTOP-NCJJCQ1J: ~$
```

可以看到第一个是正确的，剩余的结果都是大于 0 的，而加入了一个 sleep(1) 之后结果出现了 -1，显然是不正确的。sleep 会有概率导致线程调度在这个地方进行切换，从而让两个线程同时进入临界区，发生错误。

2. 双标志先检查

```
george@LAPTOP-NCJJCQ1J: ~$ vim p2_t.c
george@LAPTOP-NCJJCQ1J: ~/os/exp4$ g++ p2_t.c -l pthread -o p2_t
p2_t.c: In function 'void* print1(void*)':
p2_t.c:20:1: warning: no return statement in function returning non-void [-Wreturn-type]
    20 | }
        | ^
p2_t.c: In function 'void* print2(void*)':
p2_t.c:33:1: warning: no return statement in function returning non-void [-Wreturn-type]
    33 | }
        | ^
george@LAPTOP-NCJJCQ1J: ~/os/exp4$ ./p2_t
remain tickets: 9
remain tickets: 8
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
The End...
```

```
george@LAPTOP-NCJJCQ1J: ~$ vim p2_f.c
george@LAPTOP-NCJJCQ1J: ~/os/exp4$ g++ p2_f.c -l pthread -o p2_f
p2_f.c: In function 'void* print1(void*)':
p2_f.c:21:1: warning: no return statement in function returning non-void [-Wreturn-type]
    21 | }
        | ^
p2_f.c: In function 'void* print2(void*)':
p2_f.c:35:1: warning: no return statement in function returning non-void [-Wreturn-type]
    35 | }
        | ^
george@LAPTOP-NCJJCQ1J: ~/os/exp4$ ./p2_f
remain tickets: 9
remain tickets: 8
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
remain tickets: -1
The End...
```

可以看到第一个图结果是正确的，第二个图最后的结果发生了错误。这个错误是在我们在下图红框处加入 sleep 时得到的，这也符合实验 PPT 算法二的失效示意图，主要原因是这个算法是无法保证互斥性的。

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

//因为pthread 库非Linux 默认库，
// 编译时使用 g++ *.c -l pthread
static int tickets = 10;
bool flag[2] = {false, false};

void * print1(void* args)
{
    while(flag[1]);
    sleep(1);
    flag[0]=true;
    while(tickets>0)
    {
        tickets--;
        printf("remain tickets: %d\n", tickets);
    }
    flag[0]=false;
}

void* print2(void * args)
{
    while(flag[0]);
    sleep(1);
    flag[1]=true;
    while(tickets>0)
    {
```

3. 双标志后检查

```
george@LAPTOP-NCJJCQ1J: ~$ vim p3.c
george@LAPTOP-NCJJCQ1J: ~$ g++ p3.c -l pthread -o p3
p3.c: In function 'void* print1(void*)':
p3.c:23:1: warning: no return statement in function returning non-void [-Wreturn-type]
23 | }
   | ^
p3.c: In function 'void* print2(void*)':
p3.c:36:1: warning: no return statement in function returning non-void [-Wreturn-type]
36 | }
   | ^
george@LAPTOP-NCJJCQ1J: ~$ ./p3
remain tickets: 9
remain tickets: 8
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
The End...
george@LAPTOP-NCJJCQ1J: ~$ vim p3.c
george@LAPTOP-NCJJCQ1J: ~$ g++ p3.c -l pthread -o p3
p3.c: In function 'void* print1(void*)':
p3.c:23:1: warning: no return statement in function returning non-void [-Wreturn-type]
23 | }
   | ^
p3.c: In function 'void* print2(void*)':
p3.c:36:1: warning: no return statement in function returning non-void [-Wreturn-type]
36 | }
   | ^
george@LAPTOP-NCJJCQ1J: ~$ ./p3
^C
george@LAPTOP-NCJJCQ1J: ~$
```

第一个运行的是正常情况，还是可以看到是正常运行的，但是如果将最开始的 flag 设置为两个 true，那么程序就会死锁，导致直接卡住。

4. Peterson 算法

```
george@LAPTOP-NCJJCQ1J: ~$ vim peterson.cpp
george@LAPTOP-NCJJCQ1J: ~$ g++ peterson.cpp -l pthread -o peterson
peterson.cpp: In function 'void* print1(void*)':
peterson.cpp:26:1: warning: no return statement in function returning non-void [-Wreturn-type]
26 | }
   | ^
peterson.cpp: In function 'void* print2(void*)':
peterson.cpp:40:1: warning: no return statement in function returning non-void [-Wreturn-type]
40 | }
   | ^
george@LAPTOP-NCJJCQ1J: ~$ ./peterson
remain tickets: 9
remain tickets: 8
remain tickets: 7
remain tickets: 6
remain tickets: 5
remain tickets: 4
remain tickets: 3
remain tickets: 2
remain tickets: 1
remain tickets: 0
The End...
```

经过多组的尝试把 sleep 函数加入到不同地方，Peterson 算法得出的结果都是正确的，可以看出 Peterson 算法的鲁棒性还是很不错的。

实验 4.2 Linux 进程同步

一、实验目的

1. 加强对进程同步和互斥的理解，学会使用信号量解决资源共享问题。
2. 熟悉 Linux 进程同步原语。
3. 握信号量 wait/signal 原语的使用方法，理解信号量的定义、赋初值及 wait/signal 操作

二、实验内容

1. 任务描述

编写程序,使用 Linux 操作系统中的信号量机制模拟实现生产者-消费者问题。设有一个生产者和一个消费者,缓冲区可以存放产品,生产者不断生成产品放入缓冲区,消费者不断从缓冲区中取出产品,消费产品。

2. 实验方案

- 使用两个线程来模拟生产者和消费者
- 使用 pthread 库提供的线程操作, 需要包含头文件 pthread.h
- 使用 POSIX 的无名信号量机制, 需要包含头文件 semaphore.h

3. 实验说明

- 向缓冲区写产品

```
buffer=(char *)malloc(MAX); //给缓冲区分配内存空间
```

```
fgets(buffer,MAX,stdin); //输入产品至缓冲区
```

- 从缓冲区读产品

```
printf("read product from buffer:%s",buffer); //从缓冲区取出产品
```

```
memset(buffer,0,MAX); //清空缓冲区
```

- 线程创建、等待线程结束

```
ret=pthread_create(&id_producer,NULL,producer,NULL); //创建生产者线程
```

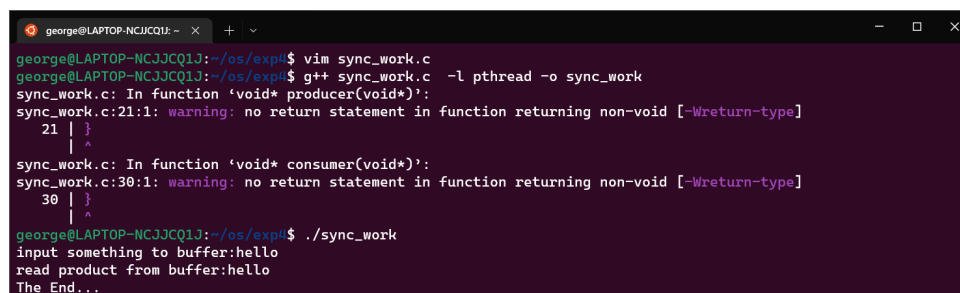
```
ret=pthread_create(&id_consumer,NULL,consumer,NULL); //创建消费者线程
```

```
pthread_join(id_producer,NULL); //等待生产者线程
```

```
pthread_join(id_consumer,NULL); //等待消费者线程结束
```

三、实验记录

实验结果



```
george@LAPTOP-NCJJCQ1J: ~/os/exp4$ vim sync_work.c
george@LAPTOP-NCJJCQ1J:~/os/exp4$ g++ sync_work.c -l pthread -o sync_work
sync_work.c: In function 'void* producer(void*)':
sync_work.c:21:1: warning: no return statement in function returning non-void [-Wreturn-type]
 21 | }
    | ^
sync_work.c: In function 'void* consumer(void*)':
sync_work.c:30:1: warning: no return statement in function returning non-void [-Wreturn-type]
 30 | }
    | ^
george@LAPTOP-NCJJCQ1J:~/os/exp4$ ./sync_work
input something to buffer:hello
read product from buffer:hello
The End...
```

可以看到程序正常的进行了缓存和取出的操作。