《操作系统原理》实验报告

姓名	杜宇晗	学号	U202112151	专业班级	信安 2104	时间	2023.11.28

一、实验目的

1）理解进程/线程的概念和应用编程过程；

2）理解进程/线程的同步机制和应用编程；

3）掌握和推广国产操作系统（推荐银河麒麟或优麒麟，建议）

二、实验内容

1）在 Linux/Windows 下创建 2 个线程 A 和 B，循环输出数据或字符串。

2）在 Liunx 下创建（fork）一个子进程，实验 wait/exit 函数

3）在 Windows/Linux 下，利用线程实现并发画圆画方。

4）在 Windows 或 Linux 下利用线程实现“生产者-消费者”同步控制

5）在 Linux 下利用信号机制(signal)实现进程通信

6）在 Windows 或 Linux 下模拟哲学家就餐，提供死锁和非死锁解法。

7）研读 Linux 内核并用 printk 调试进程创建和调度策略的相关信息。

三、实验环境和核心代码

3.1 创建一个子进程，实验 wait/exit 函数

实验环境：VMware Workstation Pro 17

Ubuntu 20.04

内核版本：5.15.0-89-generic

编辑工具：gedit

创建代码如下，先让子进程打印提示语句，进程号和父进程号，然后休眠五秒，父进程同时打印提示语句，进程号，休眠两秒，父进程先结束，然后子进程休眠完后再打印进程号和父进程号

Open process.c ~/code Save

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    pid_t pid;
    char *message;
    int n;
    pid = fork();
    if(pid < 0)
    {
        perror("fork failed");
        exit(1);
    }
    if(pid == 0)
    {
        printf("This is the child process. My PID is: %d. My PPID is: %d.\n", getpid(), getppid());
               sleep(5);
               printf("This is the child process. My PID is: %d. My PPID is: %d.\n", getpid(), getppid());
    }
    else
    {
        printf("This is the parent process. My PID is %d.\n", getpid());
               sleep(2);
    }
    return 0;
}
```

编译后运行

3.2 实现“生产者-消费者”同步控制

提示 1：使用数组（10 个元素）代替缓冲区。2 个输入线程产生产品（随机数）存到数组中；3 个输出线程从数组中取数输出。

提示 2：Linux 使用互斥锁对象和轻量级信号量对象，主要函数： sem_wait( )，sem_post( )，pthread_mutex_lock( )，pthread_mutex_unlock( )

提示 3：生产者 1 的数据：1000-1999（每个数据随机间隔 100ms-1s），生产者 2 的数据：2000-2999（每个数据随机间隔 100ms-1s）

提示 4：消费者每休眠 100ms-1s 的随机时间消费一个数据。

提示 5：屏幕打印（或日志文件记录）每个数据的生产和消费记录。

编写 pc.c 代码，如下

```
#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>
```

```
#define BufferSize 10 // Size of the buffer
#define countOfProducer 2
#define countOfConsumer 3

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize]={0};

pthread_mutex_t mutex;

void *producer(void *pno)
{
    int item;
    while(1){
        item = rand()%1000+1000*(*((int *)pno)); // Produce an random item
        sleep(1);
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Produce Item %d at %d\n", *((int
*)pno),buffer[in],in);
        in = (in+1)%BufferSize;
```

```
            pthread_mutex_unlock(&mutex);
            sem_post(&full);
        }
    }
    void *consumer(void *cno)
    {
        while(1){
            sem_wait(&full);
            pthread_mutex_lock(&mutex);
            int item = buffer[out];
            printf("Consumer %d: Consume Item %d from %d\n",*((int *)cno),item,
out);
            out = (out+1)%BufferSize;
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
            sleep(2);
        }
    }

    int main()
    {

        pthread_t pro[countOfProducer],con[countOfConsumer];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty,0,BufferSize);
```

```
    sem_init(&full,0,0);

    int a[3] = {1,2,3}; //Just used for numbering the producer and consumer

    for(int i = 0; i < countOfProducer; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(int i = 0; i < countOfConsumer; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }

    for(int i = 0; i < countOfProducer; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < countOfConsumer; i++) {
        pthread_join(con[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;

}
```

编译后运行 pc

3.3 模拟哲学家就餐,提供解法

提示 1：同时提供提供可能会带来死锁的解法和不可能死锁的解法。

提示 2：可能会带来死锁的解法参见课件。

Linux 尝试使用互斥锁(pthread_mutex_lock, pthread_mutex_unlock)

提示 3：完全不可能产生死锁的解法，例如：尝试拿取两只筷子，两只都能拿则拿，否则都不拿。

Linux 尝试使用互斥锁 pthread_mutex_lock，pthread_mutex_trylock 等函数。

提示 4：[可选]图形界面显示哲学家取筷，吃饭，放筷，思考等状态。

提示 5：为增强随机性，各状态间维持 100ms-500ms 内的随机时长。

阻塞调用

pthread_mutex_lock(&mtx);

这个操作是阻塞调用的，也就是说，如果这个锁此时正在被其它线程占用，那么 pthread_mutex_lock() 调用会进入到这个锁的排队队列中，并会进入阻塞状态，直到拿到锁之后才会返回。

非阻塞调用

如果不想阻塞，而是想尝试获取一下，如果锁被占用咱就不用，如果没被占用那就用，这该怎么实现呢？可以使用 pthread_mutex_trylock() 函数。这个函数和 pthread_mutex_lock() 用法一样，只不过当请求的锁正在被占用的时候，不会进入阻塞状态，而是立刻返回，并返回一个错误代码 EBUSY，意思是说，有其它线程正在使用这个锁。

非死锁(使用信号量)

这里注意设置了 room，因为 5 个人同时进入会可能产生死锁，所以限定最多 4 个人同时进入并是未吃完的状态。

创建 phi.c，输入以下内容

```
#include<stdio.h>

#include<stdlib.h>
```

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

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
    int i,a[5];
    pthread_t tid[5];

    sem_init(&room,0,4);

    for(i=0;i<5;i++)
        sem_init(&chopstick[i],0,1);

    for(i=0;i<5;i++){
        a[i]=i;
        pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
    }
    for(i=0;i<5;i++)
        pthread_join(tid[i],NULL);
```

```
}

void * philosopher(void * num)
{
    int phil=*(int *)num;

    sem_wait(&room);
    printf("\nPhilosopher %d has entered room",phil);
    sem_wait(&chopstick[phil]);
    sem_wait(&chopstick[(phil+1)%5]);

    eat(phil);
    sleep(1);
    printf("\nPhilosopher %d has finished eating",phil);

    sem_post(&chopstick[(phil+1)%5]);
    sem_post(&chopstick[phil]);
    sem_post(&room);
}

void eat(int phil)
{
    printf("\nPhilosopher %d is eating",phil);
}
```

编译后运行 phi

死锁(使用阻塞调用的互斥锁)

创建 dl.c，输入以下内容

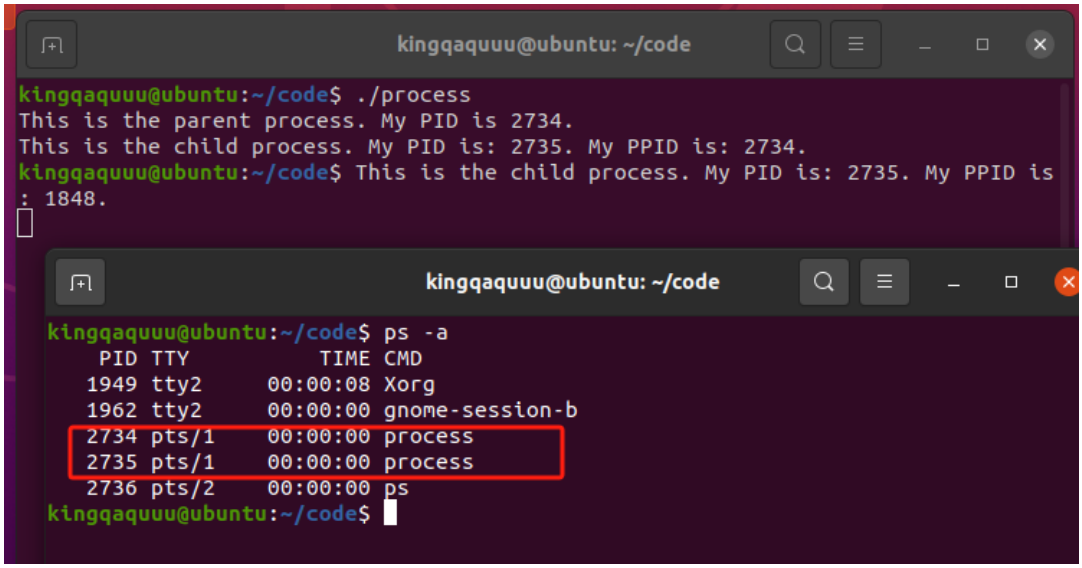
```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>

pthread_mutex_t chopstick[5];
void * philosopher(void *);
void eat(int);
int main()
{
        int i,a[5];
        pthread_t tid[5];

        for(i=0;i<5;i++)
                pthread_mutex_init(&chopstick[i], NULL);

        for(i=0;i<5;i++){
                a[i]=i;
                pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
        }
        for(i=0;i<5;i++)
                pthread_join(tid[i],NULL);
        for(i=0;i<5;i++)
                pthread_mutex_destroy(&chopstick[i]);
}

void * philosopher(void * num)
{
        int phil=*(int *)num;
        printf("\nPhilosopher %d has entered room",phil);
        pthread_mutex_lock(&chopstick[phil]);
        sleep(1);
        pthread_mutex_lock(&chopstick[(phil+1)%5]);
        eat(phil);
        sleep(2);
        printf("\nPhilosopher %d has finished eating",phil);
        pthread_mutex_unlock(&chopstick[(phil+1)%5]);
        pthread_mutex_unlock(&chopstick[phil]);
}

void eat(int phil)
{
        printf("\nPhilosopher %d is eating",phil);
}
```

编译后运行 dl

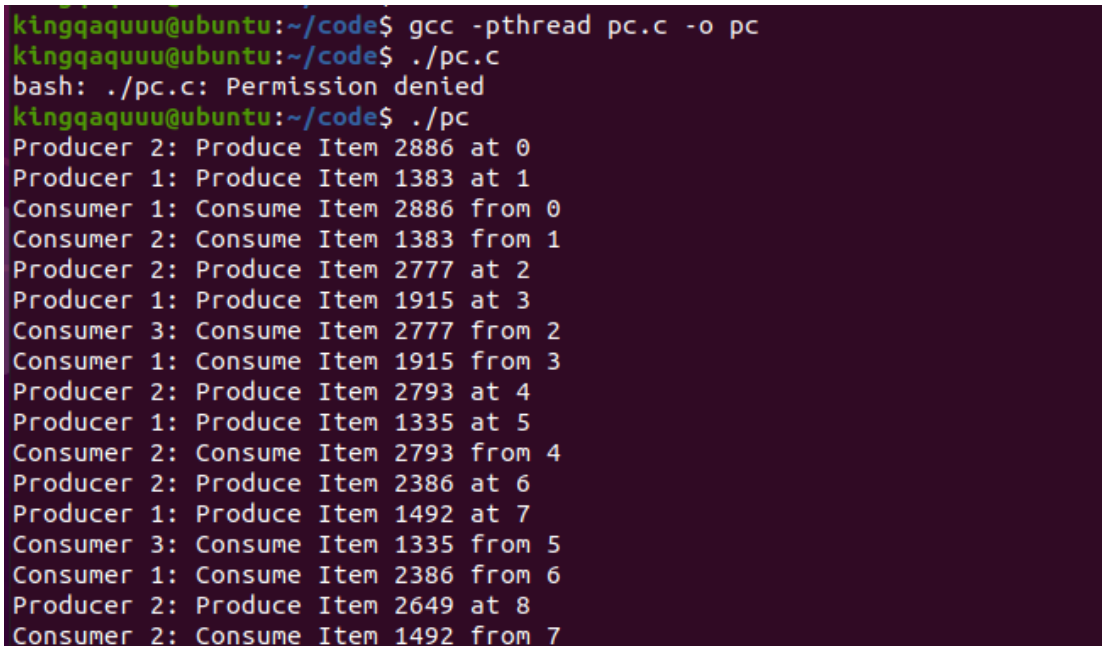
四、实验结果

4.1 创建一个子进程，实验 wait/exit 函数

可以发现使用 ps 可以看到两个进程，2734 和 2735，是父子进程，然后让父进程提前结束



4.2 实现“生产者-消费者”同步控制



```
Producer 2: Produce Item 2429 at 3
Consumer 1: Consume Item 1736 from 8
Consumer 2: Consume Item 2172 from 9
Producer 1: Produce Item 1782 at 4
Producer 2: Produce Item 2530 at 5
Consumer 3: Consume Item 1211 from 0
Producer 1: Produce Item 1862 at 6
Producer 2: Produce Item 2123 at 7
Consumer 1: Consume Item 2368 from 1
Consumer 2: Consume Item 1567 from 2
Producer 2: Produce Item 2135 at 8
Producer 1: Produce Item 1067 at 9
Consumer 3: Consume Item 2429 from 3
Producer 2: Produce Item 2929 at 0
Producer 1: Produce Item 1802 at 1
Consumer 1: Consume Item 1782 from 4
Consumer 2: Consume Item 2530 from 5
Producer 2: Produce Item 2022 at 2
Producer 1: Produce Item 1058 at 3
Consumer 3: Consume Item 1862 from 6
Producer 2: Produce Item 2069 at 4
Producer 1: Produce Item 1167 at 5
Consumer 1: Consume Item 2123 from 7
Consumer 2: Consume Item 2135 from 8
```

因为 consumer 比 producer 多一人，所以让 consumer 休眠时间长一些，否则就会刚生产出来就被消耗掉

4.3 模拟哲学家就餐,提供解法

非死锁

```
kingqaquuu@ubuntu:~/code$ gcc -pthread phi.c -o phi
kingqaquuu@ubuntu:~/code$ ./phi

Philosopher 0 has entered room
Philosopher 0 is eating
Philosopher 3 has entered room
Philosopher 3 is eating
Philosopher 2 has entered room
Philosopher 1 has entered room
Philosopher 3 has finished eating
Philosopher 2 is eating
Philosopher 4 has entered room
Philosopher 0 has finished eating
Philosopher 4 is eating
Philosopher 2 has finished eating
Philosopher 1 is eating
Philosopher 4 has finished eating
Philosopher 1 has finished eatingkingqaquuu@ubuntu:~/code$
```

死锁

```
kingqaquuu@ubuntu:~/code$ gcc -pthread dl.c -o dl
kingqaquuu@ubuntu:~/code$ ./dl

Philosopher 0 has entered room
Philosopher 1 has entered room
Philosopher 2 has entered room
Philosopher 4 has entered room
```

发生死锁

五、实验错误排查和解决方法

5.1 创建一个子进程，实验 wait/exit 函数

暂无

5.2 实现“生产者-消费者”同步控制

在编译文件时候，

```
kingqaquuu@ubuntu:~/code$ gcc pc.c -o pc
/usr/bin/ld: /tmp/ccw1ld4d.o: in function `producer':
pc.c:(.text+0x5a): undefined reference to `sem_wait'
/usr/bin/ld: pc.c:(.text+0x10a): undefined reference to `sem_post'
/usr/bin/ld: /tmp/ccw1ld4d.o: in function `consumer':
pc.c:(.text+0x12b): undefined reference to `sem_wait'
/usr/bin/ld: pc.c:(.text+0x1c4): undefined reference to `sem_post'
/usr/bin/ld: /tmp/ccw1ld4d.o: in function `main':
pc.c:(.text+0x215): undefined reference to `sem_init'
/usr/bin/ld: pc.c:(.text+0x22b): undefined reference to `sem_init'
/usr/bin/ld: pc.c:(.text+0x282): undefined reference to `pthread_create'
/usr/bin/ld: pc.c:(.text+0x2ce): undefined reference to `pthread_create'
/usr/bin/ld: pc.c:(.text+0x2f8): undefined reference to `pthread_join'
/usr/bin/ld: pc.c:(.text+0x322): undefined reference to `pthread_join'
/usr/bin/ld: pc.c:(.text+0x344): undefined reference to `sem_destroy'
/usr/bin/ld: pc.c:(.text+0x350): undefined reference to `sem_destroy'
collect2: error: ld returned 1 exit status
```

遇到了该报错，解决方法是在编译时实用-pthread 解决

5.3 模拟哲学家就餐,提供解法

暂无

六、实验参考资料和网址

（1）教学课件