《操作系统原理》实验报告

姓名	鄢湧棚	学号	U201911741	专业班级	网安 1902	时间	2022.1.6

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；

二、 实验内容

1）在 Linux 下创建一对父子进程。

2）在 Linux 下创建 2 个线程 A 和 B，循环输出数据或字符串。

3）在 Windows 下创建线程 A 和 B，循环输出数据或字符串。

4）在 Linux 下创建一对父子进程，实验 wait 同步函数。

5）在 Windows 下利用线程实现并发画圆/画方。

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

7）在 Linux 下利用信号机制实现进程通信。

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

1,6,8 必做,再选做 2。

三、 实验过程

1）在 Linux 下创建一对父子进程。

编写 c 代码如下：

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int ct,pt; //设置父子进程休眠时间
int main(int argc, char** argv){
```

```
    pid_t pid;
    switch(argv[1][0]){
    case '0':
        printf("\n 普通情况：\n");
        ct=pt=5;
        break;
    case '1':
        printf("\n 父进程先结束：\n");
        pt=2,ct=8;
        break;
    case '2':
        printf("\n 子进程先结束：\n");
        pt=8,ct=2;
        break;

    }
    pid=fork();
    if(pid==0) {
        sleep(ct);
        printf("\nChild Process: pid:%d ppid:%d\n",
            getpid(),getppid()); //输出进程 id 和父进程 id
    }
    else if(pid>0) {
        sleep(pt);
        printf("\nParent Process: pid:%d ppid:%d\n",
            getpid(),getppid());
    }
    return 0;
}
```

然后分别使用 0，1，2 三种情况得到不同输出。

2）在 Linux 下创建 2 个线程 A 和 B，循环输出数据或字符串。

编写代码如下

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

void *thread1(void *)
{
    for (int i = 1; i <= 100; i++)
    {
        printf("A:%4d\n", i);
        sleep(0.2);
    }
    return NULL;
}

void *thread2(void *)
{
    for (int j = 100; j >= 0; j--)
    {
        printf("B:%4d\n", j);
        sleep(0.2);
    }
    return NULL;
}
int main()
{
    int ret;
```

```
    pthread_t pid_A, pid_B;
    ret = pthread_create(&pid_A, NULL, thread1, NULL);
    if (ret)
        printf("Create thread1 error\n");
    ret = pthread_create(&pid_B, NULL, thread2, NULL);
    if (ret)
        printf("Create thread2 error\n");

    ret = pthread_join(pid_A, NULL);
    printf("A:%d\n", ret);
    ret = pthread_join(pid_B, NULL);
    printf("B:%d\n", ret);

    return 0;
}
```

1000 太多了，弄到 100 就行，直接编译运行。

6）在 Linux 下利用线程实现“生产者-消费者”同步控制

生产者消费者同步用到了互斥锁和同步 PV 操作，定义一个互斥锁，两个信号量 notfull 和 notempty。

对生产者和消费者的线程函数编写如下：

生产者

```
void *producer(void *)
{
    while (flag)
    {
        pthread_mutex_lock(&mutex); //加锁
        //满了就不能往里加了
        while (!space)
            pthread_cond_wait(&notfull, &mutex); //条件不满足就会
```

```
wait 阻塞自己，同时释放线程锁
        //临界区
        space--;
        items++;
        store[now] = 1;
        now++;
        printf("Producer id = %lu, add to %d\n",pthread_self(),now);
        //临界区
        pthread_cond_signal(&notempty); //将消费者所需条件置为真
        pthread_mutex_unlock(&mutex); //开锁
        sleep(0.02); //休息 20ms
    }
    pthread_exit(0);
}
```

消费者

```
void *consumer(void *)
{
    while(flag){
        pthread_mutex_lock(&mutex);
        //没有物品不能消费
        while(!items) pthread_cond_wait(&notempty,&mutex);
        //临界区
        space++;
        items--;
        store[now]=0;
        now--;
        printf("Consumer id = %lu, take %d\n",pthread_self(),now+1);
        //临界区
```

```
        pthread_cond_signal(&notfull);
        pthread_mutex_unlock(&mutex);
        sleep(0.02); //休息 20ms
    }
    pthread_exit(0);
}
```

对于总程序，设定每个线程函数在 flag 为真时循环进行，在 main 函数，即主线程中设置 0.5s 的延迟，使两个线程都运行 0.5s，观察运行结果。

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

死锁解法：

以下为哲学家的线程函数，其中，将思考抽象为 sleep(rand)的过程，将吃饭也抽象成挂起随机时间，为了使结果更明显，在拿起第一根筷子后，即为左边筷子加锁后，每个哲学家都 sleep(2)，增加产生死锁的可能性。

```
void *philosophy(void *t) //第 i 位哲学家
{
    float think;
    int i = *((int*) t);
    printf("%d\n",i);
    int ret;
    while(flag){
        think = (rand()%400 + 100)/1000;
        sleep(think);   //思考
        pthread_mutex_lock(&mutex[i]); //加锁
        sleep(2);
        pthread_mutex_lock(&mutex[(i+1)%5]);
        //等待左右筷子可用
        //吃饭
        ate[i]=1;
        printf("Philosophor %d is eating\n",i+1);
```

```
        think = (rand()%400 + 100)/1000;
        sleep(think);
        //放下筷子
        pthread_mutex_unlock(&mutex[i]);
        pthread_mutex_unlock(&mutex[(i+1)%5]);
    }
    pthread_exit(0);
}
```

非死锁解法：限制拿到筷子的人数最多为 4 个人：

```
void *philosophy(void *t) //第 i 位哲学家
{
    float think;
    int i = *((int*) t);
    printf("%d\n",i);
    int ret;
    while(flag){
        think = (rand()%400 + 100)/1000;
        sleep(think);   //思考 a
        if(chops == 4) continue;
        pthread_mutex_lock(&mutex[i]); //加锁
        pthread_mutex_lock(&chop);
        chops++;
        pthread_mutex_unlock(&chop);
        sleep(2);
        pthread_mutex_lock(&mutex[(i+1)%5]);
        //等待左右筷子可用
        //吃饭
        ate[i]=1;
```

```
        printf("Philosophor %d is eating\n",i+1);
        think = (rand()%400 + 100)/1000;
        sleep(think);
        //放下筷子
        pthread_mutex_unlock(&mutex[i]);
        pthread_mutex_unlock(&mutex[(i+1)%5]);

        pthread_mutex_lock(&chop);
        chops--;
        pthread_mutex_unlock(&chop);
    }
    pthread_exit(0);
}
```

四、 实验结果

1)在 Linux 下创建一对父子进程。

普通情况父子进程同时结束，但是父进程的运行比子进程快了一个 fork 函数的时间差，所以父进程先输出。

```
yyp@yyp-virtual-machine:~/Document/C$ ./thread 0
普通情况:

Parent Process: pid:5451 ppid:5437
Child Process: pid:5452 ppid:5451
```

父进程先结束，父进程结束后终端会返回，但是此时子进程还未结束，所以在终端返回后，子进程会在返回后输出。

```
yyp@yyp-virtual-machine:~/Document/C$ ./thread 1
父进程先结束:

Parent Process: pid:5459 ppid:5437
yyp@yyp-virtual-machine:~/Document/C$
Child Process: pid:5460 ppid:1112
```

子进程先结束，则子进程先输出，然后父进程输出后返回到终端。

```
yyp@yyp-virtual-machine:~/Document/C$ ./thread 2

子进程先结束:

Child Process: pid:5463 ppid:5462

Parent Process: pid:5462 ppid:5437
```

2)在 Linux 下创建 2 个线程 A 和 B，循环输出数据或字符串。

```
yyp@yyp-virtual-machine:~/Document/C$ ./thread2
A:   1
B: 100
B:  99
A:   2
A:   3
B:  98
A:   4
A:   5
B:  97
B:  96
A:   6
A:   7
B:  95
A:   8
B:  94
B:  93
A:   9
A:  10
B:  92
```

可以看到 AB 两个线程基本上是同步输出的。

6）在 Linux 下利用线程实现“生产者-消费者”同步控制

```
Producer id = 139744666265152, add to 2
Consumer id = 139744750192192, take 2
Producer id = 139744683050560, add to 2
Producer id = 139744666265152, add to 3
Consumer id = 139744750192192, take 3
Consumer id = 139744766977600, take 2
Consumer id = 139744758584896, take 1
Producer id = 139744683050560, add to 1
Producer id = 139744523654720, add to 2
Producer id = 139744674657856, add to 3
Producer id = 139744641087040, add to 4
Consumer id = 139744758584896, take 4
Consumer id = 139744699835968, take 3
Consumer id = 139744733406784, take 2
Consumer id = 139744766977600, take 1
Producer id = 139744641087040, add to 1
Producer id = 139744624301632, add to 2
Consumer id = 139744725014080, take 2
Consumer id = 139744758584896, take 1
Producer id = 139744683050560, add to 1
Producer id = 139744641087040, add to 2
Producer id = 139744545601088, add to 3
Consumer id = 139744758584896, take 3
Consumer id = 139744699835968, take 2
Producer id = 139744683050560, add to 2
Producer id = 139744641087040, add to 3
Consumer id = 139744716621376, take 3
Producer id = 139744523654720, add to 3
Producer id = 139744624301632, add to 4
Producer id = 139744683050560, add to 5
Producer id = 139744641087040, add to 6
Producer id = 139744632694336, add to 7
Consumer id = 139744691443264, take 7
Producer id = 139744674657856, add to 7
Producer id = 139744666265152, add to 8
Producer id = 139744666265152, add to 9
Producer id = 139744641087040, add to 10
Consumer id = 139744708228672, take 10
Producer id = 139744657872448, add to 10
Consumer id = 139744708228672, take 10
```

可以看到生产者和消费者的同步关系，始终不会超过最大存储数 10，在生产者生产 10 个时，会阻塞先由消费者消费再生产。

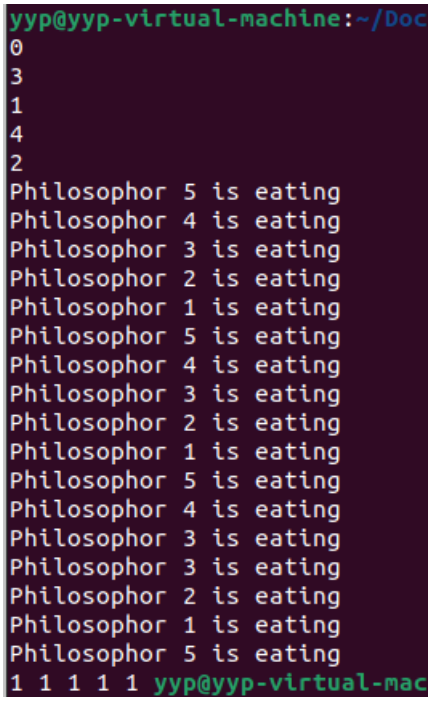
8）在 Windows 或 Linux 下模拟哲学家就餐，提供死锁和非死锁解法。

死锁解法，为了体现死锁，在哲学家的函数中，取了第一根筷子后会 sleep 两秒，这样就一定会发生死锁。

```
0
1
2
3
4
```

可以看到，在输出各自的编号后，程序就不再运行了。

非死锁解法，可以看到，程序能正常运行完，并且每个哲学家都吃到了饭。



五、 实验错误排查和解决方法

1）在 Linux 下创建一对父子进程。

没什么问题

2)在 Linux 下创建 2 个线程 A 和 B，循环输出数据或字符串。

输出 1000 个太多了，改成输出 100 个，没什么问题

6）在 Linux 下利用线程实现“生产者-消费者”同步控制

最开始忘记设置是否已满的信号量，导致内存溢出，后来加了就可以了

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

死锁现象很难触发，解决方法是在拿起第一根筷子后先 sleep，这样就一定会触发死锁。

六、 实验参考资料和网址

实验 ppt

https://www.cnblogs.com/x_wukong/p/5671137.html

https://blog.csdn.net/l_searcing/article/details/83649962