**《操作系统原理》实验报告**

**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

**三、实验过程**

**3.1 Linux下循环输出数据**

在Linux下利用库函数< pthread.h >进行程序的编写。在main函数中用函数pthread\_create创建两个线程。之后使用函数pthread\_join阻塞主线程的执行，直至目标线程tid1和tid2执行结束，实现tid1和tid2的同步。



**3.2 Linux下fork一个子进程**

效果一：父进程不用wait函数，先于子进程结束。

首先用fork函数创建一个子进程，由于fork函数的返回值特性，利用返回值区分子进程和父进程。在父进程中，打印父进程和子进程的PID，休眠五秒，同时利用在命令行中利用ps命令查看两个进程的PID。在子进程中，利用死循环，使父进程先于子进程结束。



效果二：父进程用wait函数，后于子进程结束。

首先用fork函数创建一个子进程，与上一程序思路相似，只是在父进程中不再休眠，而是利用wait() 函数等待子进程执行完毕，并打印出子进程返回的值。在子进程中，利用exit()函数退出该进程，并传递返回值，此处设定返回值为0。



**3.3 Windows下利用线程并发画圆画方**

画圆画方需要用到库graphics.h中的函数，进入网站easyX进行头文件的安装下载。利用函数putpixel进行画点，之后连成线。如下代码所示，画圆时，设定像素点共720个，每个函数执行间隔5ms，依据圆的形状坐标对圆图形进行绘制。



在Windows进行线程控制时，主要利用函数CreateThread与WaitForSingleObject，与Linux下的并发过程类似，创建线程后利用函数阻塞目前的主线程，等待圆线程和方线程结束。



**3.4 Linux下实现“生产者-消费者”同步控制**

首先基于要求，定义生产者的数量为常量 2，消费者的数量为常量 3，同时设置大小为10 的数组(缓冲区)用于存放“产品”；fullFlag 和 emptyFlag 声明为初始信号量， 在 main 函数中通过 init 函数分别初始化为 10 和 0，分别表示当前数组中的空位和已存入的元素数量。 结合相应的 pv 保证：生产者在数组满时会阻塞等待消费者进行消费操作，而非继续生产； 消费者在数组空时会阻塞等待生产者进行生产操作，而非继续消费；声明一个大小为生产者与消费者数量之和的数组 tid，用于创建线程时存储指向线程标识符的指针；声明一个互斥锁对象 mutex，实现任一生产者和生产者、生产者和消费者、消费者之间对于数组访问操作的互斥关系；ProPos 和 ConPos 两个变量分别用于标记当前生产者和消费者操作的数组位置。



对于生产者线程，使用while(1)循环表明生产者不断地重复生产指定范围数的操作。下面代码列出不断循环的操作。在每次要开始真正的生产操作前，需要调用sem\_wait(&fullFlag)，保证不会出现数组满依然继续生产，之后调用pthread\_mutex\_lock(&mutex)保证操作的互斥；之后根据函数传入的信息，输出对应生产者范围内的物品号码，及存放的数组下标位置，并更新当前生产者完成操作的位置（ProPos），注意所有生产者共用变量ConPos；最后用函数pthread\_mutex\_unlock执行相应的解锁操作，用函数sem\_post实现v操作。



对于消费者线程，在每次要开始真正的消费操作前，调用 sem\_wait(&emptyFlag)，保证不会出现数组空依然继续消费，之后调用 pthread\_mutex\_lock(&mutex)保证操作的互斥；之后输出提示信息：包括当前消费者的编号、消费的数组元素值、消费的数组元素下标。更新当前消费者完成操作的位置(ConPos)，最后用函数pthread\_mutex\_unlock执行相应的解锁操作，用函数sem\_post实现v操作。



对于线程控制，在main函数中完成。首先完成互斥量的初始化，以及信号量的初始化；之后创建一个线程池子，分别为 2 个消费者、3 个生产者创建线程， 这里要特别注意pthread\_create函数的参数及参数的含义，以免出现问题；调用 5 次 pthread\_join()函数以阻塞的方式等待所有消费者和生产者线程结束；最后释放所有互斥量和信号量。



**四、实验结果**

**4.1 Linux下循环输出数据**

编译运行之后，观察到标识为 B 的线程从 1000 开始递减 1 进行输出，而标识为 A 的线 程从 1 开始递增 1 进行输出，可以看到两个线程交替输出正确的预期结果，如图4-1所示。

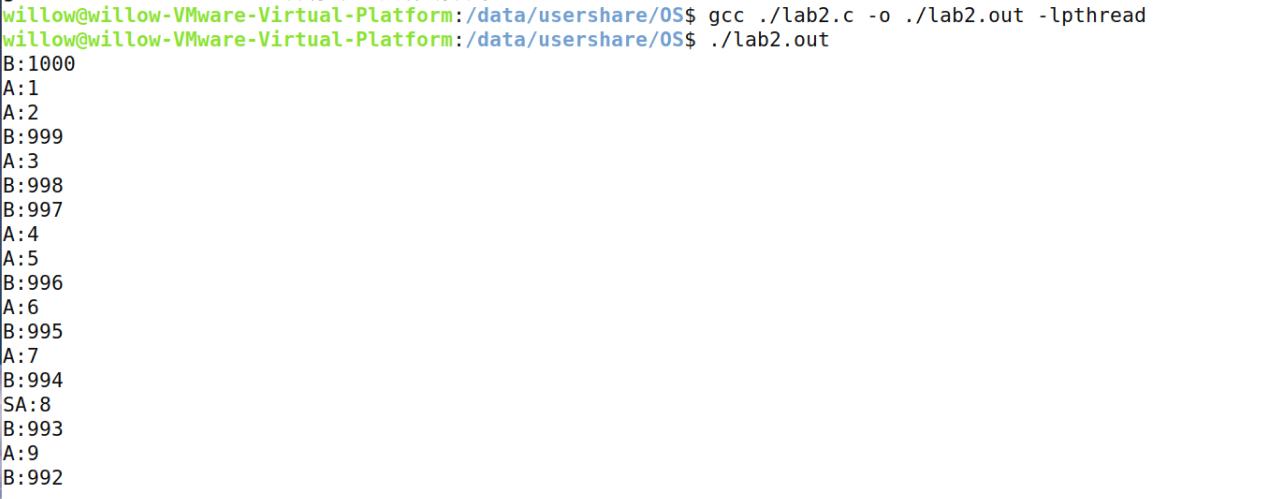


图4-1 循环输出数据运行结果

**4.2 Linux下fork一个子进程**

运行实现效果一的程序，可以看到输出部分。对于子进程， pid 为 8487，其父进程 pid 为 8486；对于父进程， pid 为 8486 (其父进程 pid 为 2197)；由于该程序中控制父进程在子进程后结束，因此子进程的父进程 pid 应当与创建该子进程的进程 pid 相同，如图4-2，验证了这一结果。

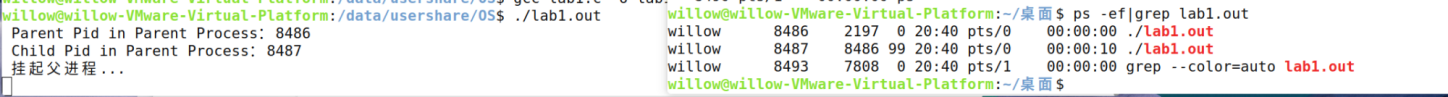


图4-2 效果一运行结果

运行实现效果二的程序，可以看到，在父进程中打印出父进程PID为8977，然后进入阻塞开始等待子进程的结束；子进程打印出自己的PID为8978，睡眠五秒后结束；最后父进程结束。如图4-3所示。

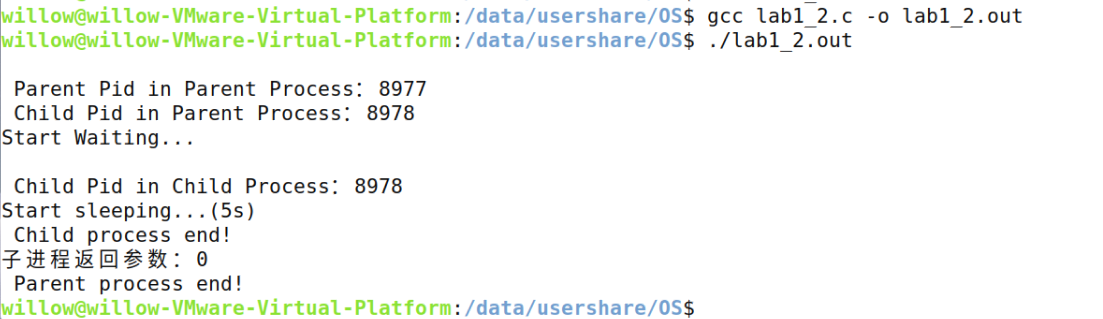


图4-3 效果二运行结果

**4.3 Windows下利用线程并发画圆画方**

如下图所示，该程序可以同时画圆画方，左侧为正方形线程画出的图案，右侧为圆形线程画出的图案，两者由于像素点个数相同，所以可以同时结束，如图4-4所示。

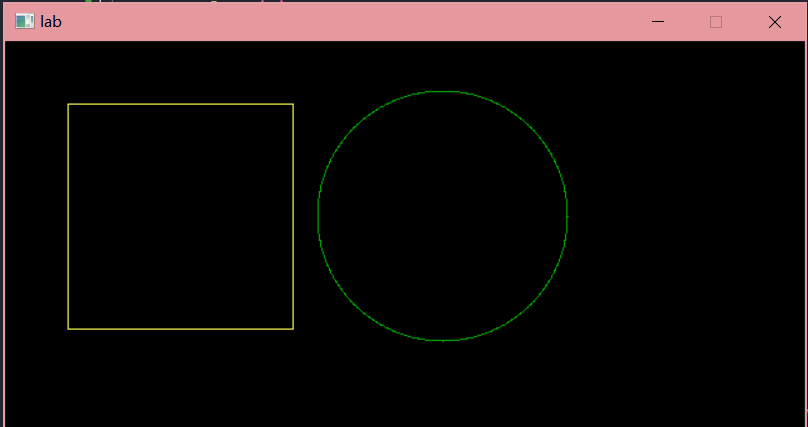


图4-4 同时画圆画方运行结果

**4.4 Linux下实现“生产者-消费者”同步控制**

编译运行，查看运行结果，可以看到在终端的输出中 2 个生产者依次生产数据，而 3 个消费者依次消费数据，并且都能够清楚地指出生产数据/消费数据的值和数组下标位置，依次查看对比，发现对于任意数据也都满足先生产后消费的规律，满足要求，如图4-5所示。



图4-5 “生产者-消费者”运行结果

**五、实验错误排查和解决方法**

**5.1 Linux下循环输出数据**

Q：对库pthread.h里的函数不熟悉。

A：查阅相关资料，学习各个参数的表达的意思。

对函数int pthread\_create ( pthread\_t \* thread , const pthread\_attr\_t \* attr ,void \*(\* start\_routine ) (void \* ),void \* arg )—— thread指向的地址存储创建的线程号；attr是一个 pthread\_attr\_t 结构体的指针，用来在线程创建的时候指定新线程的属性。如果在创建线程时，这个参数指定为 NULL, 那么就会使用默认属性。新线程通过调用start\_routine（）开始执行; arg作为start\_routine（）的唯一参数传递。

对函数int pthread\_join(pthread\_t thread, void \*\* retval) —— thread用于指定接收哪个线程的返回值；第二参数表示接收到的返回值，如果 thread 线程没有返回值，或不需要接收 thread 线程的返回值，可以将 retval 参数置为 NULL。

**5.2 Linux下fork一个子进程**

Q：sleep(5\*1000)等待时间过长终端无反应。

A：与Windows系统不同，在Linux系统中，使用的是gcc的库，sleep是以秒为单位的，所以需要休眠五秒时应该使用sleep( 5 )。

**5.3 Windows下利用线程并发画圆画方**

Q：不会使用画图函数库？

A：网上查阅资料后，学习了相关的库函数之后，选择使用函数putpixel进行绘制。对函数void putpixel(int x,int y,COLORREF color)中，前两参数为点的横纵坐标，第三参数为点的颜色。

Q：绘制圆的公式？

A：以坐标点( cos(-pi / 2 + ( (i \* pi) / 360) ), sin( -pi / 2 + ((i \* pi) / 360 )为基础，进行变换。

**5.4 Linux下实现“生产者-消费者”同步控制**

Q：rand函数每次产生的随机值相同？

A：在C语言中，rand函数可以用来产生随机数，但并不是真正意义上的随机数。它是返回介于 0 和 RAND\_ MAX 之间的伪随机整数。这个数字是由一个算法生成的，该算法每次调用它时都返回一个显然不相关的数字序列。该算法是使用一个种子来生成序列，当计算机正常开机后，这个种子的值是固定的，因此产生的伪随机整数也是固定的，除非你为了改变这个值破坏了系统。为了初始化的值不同，C语言提供了srand函数。

rand函数每次调用前都会查询是否调用过srand(seed)，是否给seed设定了一个值，如果没有，种子的值就默认为1，直接用 1 来初始化种子，那生成的随机数每次就会重复，为了防止生成的随机数重复，一般使用时间戳作为时间种子，采用系统时间来初始化，使用time函数来获得系统时间，它的返回值为从 00:00:00 GMT, January 1, 1970 到现在所持续的秒数，然后将time\_t型数据转化为(unsigned)型再传给srand函数。

所以要产生真正的随机数需要在使用rand函数前先用srand随机化种子。

**六、实验参考资料和网址**

**（1）教学课件**

**（2）URL：https://blog.csdn.net/weixin\_44518102/article/details/124622003**

**（3）URL：https://blog.csdn.net/m0\_47988201/article/details/116332597**