《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 刘逸凡 | 学号 | U202012155 | 专业班级 | 网安2004 | 时间 | 2022.11.29 |

**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。

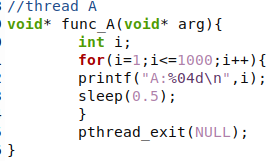
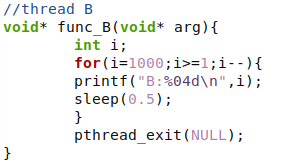
 7）研读Linux内核并用printk调试进程创建和调度策略的相关信息

**三、实验过程**

**3.1 双线程循环输出数据**

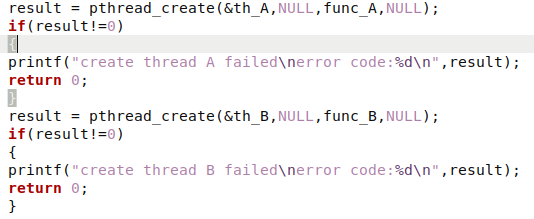
1）定义线程函数

分别定义线程A与线程B要执行的函数，注意格式化输出

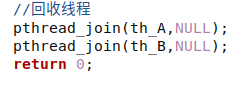
 

2）创建线程

通过pthread\_create创建线程



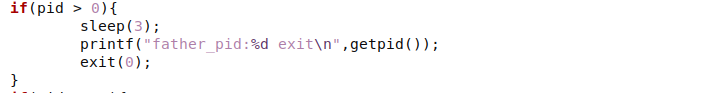
3）回收线程



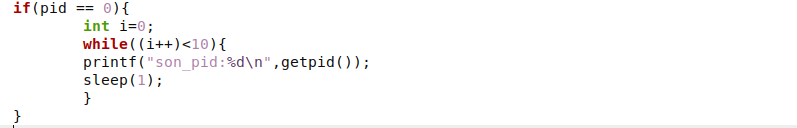
**3.2 实验wait/exit函数**

1）父进程不用wait函数

在通过fork()创建子进程后，父进程使用exit()结束进程

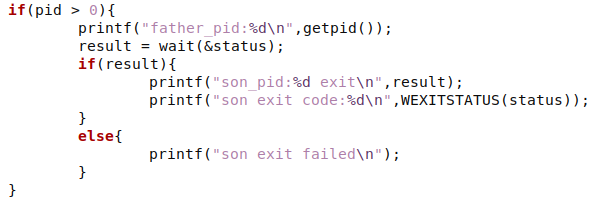


子进程则进入时间较长的循环

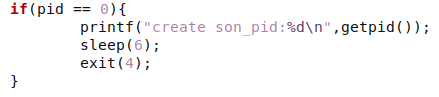


2）父进程使用wait函数

通过fork函数创建子进程后，父进程使用wait函数等待回收子进程，并且使用WEXITSTATUS函数读取子进程的退出码



子进程中，休眠一定时间后用exit函数退出，并且退出码设为4



**3.3 生产者-消费者同步控制**

1）定义信号量

定义了三个信号量SPACE、DATA、MUTEX，其中SPACE表示剩余空白数据的多少，DATA表示已有数据的多少，MUTEX则用于对数据读写的互斥



SPACE的初始值为10，DATA初始值为0，表示所有数据块均未有数据，MUTEX初始值为1，表示同一时间只能有一个生产者/消费者进行读/写



1. 生产者线程

生产者维护了一个数组producer\_order，其中保存了当前还是空白的数据块的序号，以及一个整型变量space\_size，保存了当前空白数据块的数量，两者初始值如图所示



生产者每次在空白数据块序号中随机抽取一个数据块向其中填入数据，然后更新producer\_order与space\_size



1. 消费者线程

与生产者类似，消费者维护了一个数组consume\_order，其中保存了当前已有数据的数据块的序号，以及一个整型变量data\_size，保存了当前已有数据的数据块的数量，两者初始值如图所示



消费者每次在已有数据的数据块中随机抽取一块读出数据并将数据块数据置零，然后更新consume\_order与data\_size



**3.4 进程通信**

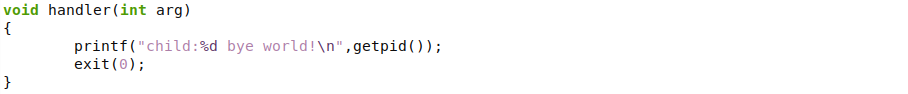
1）共享内存的实现

定义了一个整型变量shmid，一个指针型变量share\_mem，通过shmget函数创建共享内存并将id保存至shmid中，其中参数IPC\_CREATE | IPC\_EXCL保证每次创建的都是新的一块内存，S\_IRUSR | S\_IWUSR表示可以对共享内存进行读写



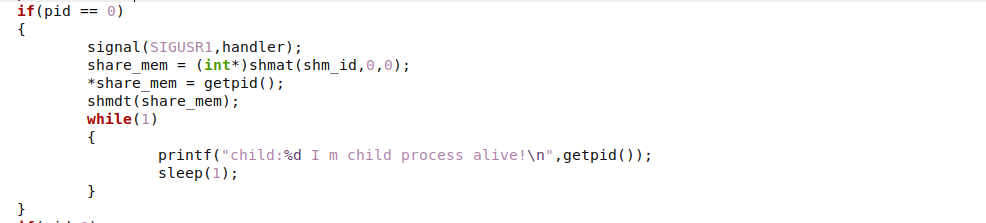
2）子进程

子进程通过signal函数接收父进程发来的信号SIGUSR1（用户自定义信号），收到信号则执行handler函数，即结束进程



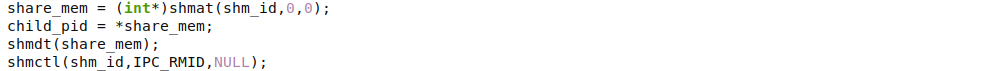


子进程通过shmat函数获得共享内存的指针保存至share\_mem中，然后通过share\_mem将getpid()即子进程的pid保存至共享内存中并断开共享内存连接，最后进入一个死循环打印信息提示子进程一致存活

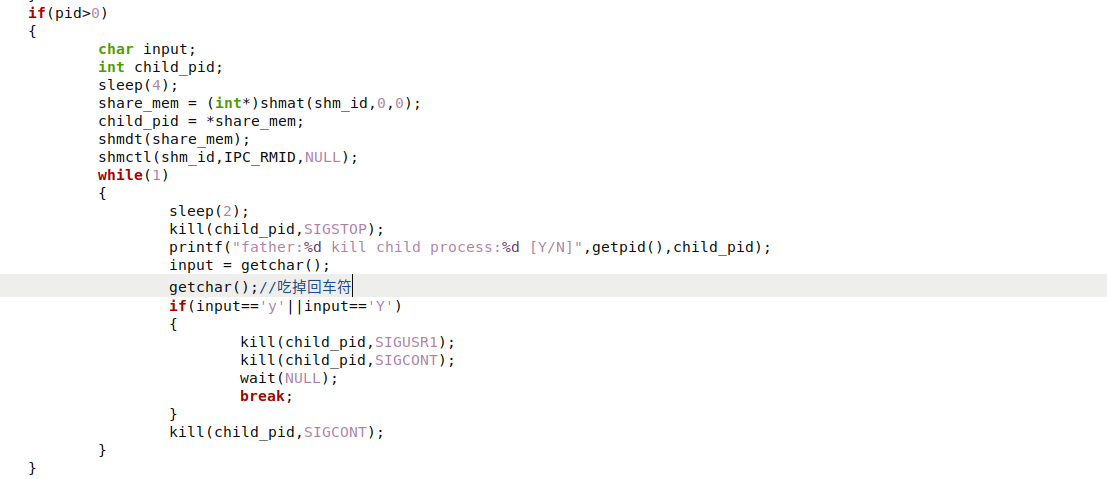


3）父进程

与子进程类似，首先通过shmat获取共享内存的指针，然后从中读取到子进程的pid后则断开共享内存连接并且关闭共享内存



接着同样进入死循环，每隔2秒询问一次是否关闭子进程，同时此时发送SIGSTOP信号暂时暂停子进程，防止父进程与子进程打印的信息混在一起，若同意则发送SIGUSR1信号关闭子进程同时发送SIGCONT信号继续子进程，若不同意则仅发送SIGCONT信号使子进程继续



**四、实验结果**

**4.1 双线程循环输出数据**

实验结果如下图所示



因为A和B并不是严格意义上的轮流执行，所以会出现A先于B打印完或B先于A打印完的情况

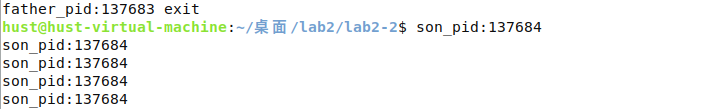
 

**4.2 实验wait/exit函数**

1）首先是父进程不用wait函数，即在父进程结束后子进程继续执行；在刚开始父子进程都存活时，使用ps -au可以看到父子进程的进程号



在父进程退出后，子进程依然在运行



用ps命令可以看到只有子进程的进程号



2）父进程使用wait函数等待子进程结束；首先在程序开始父子进程存活时



使用ps命令打印进程信息，可以看到父子进程两者的进程号

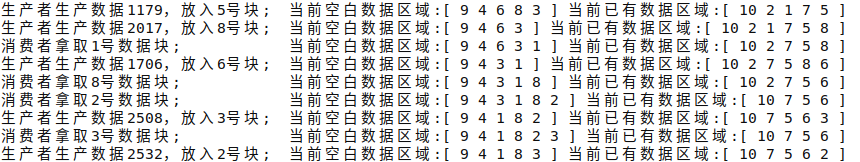


待子进程用exit函数结束并返回退出码后，父进程收到退出码并打印出相关信息后退出



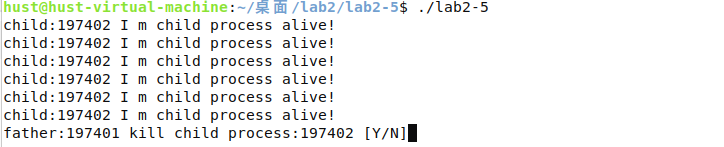
**4.3 生产者-消费者同步控制**

实验结果如下图所示，其中打印出了数据为空的数据块序号以及已有数据的数据块序号

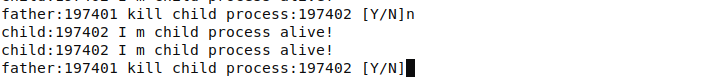


**4.4 进程通信**

首先待父进程休眠一段时间后提示是否结束子进程，此时子进程被暂停暂时不会打印信息



输入n即不同意后子进程将继续运行且父进程重新进入休眠直至下一次提示



此时输入y即同意结束子进程后，子进程将结束并打印相关信息，且父进程同样结束



**五、实验错误排查和解决方法**

**5.1 双线程循环输出数据**

1）在涉及到线程时，编译需要加一句-lpthread参数，即gcc -o xxx xxx.c -lpthread

2）一开始在线程里打印信息时是如下图打印方式，即有两句或以上printf语句



这样会导致输出错位



需要利用信号量进行互斥或只用一句printf打印

**5.2 实验wait/exit函数**

子进程最好进入较长的循环而不是死循环，因为变成孤儿进程后死循环将持续打印信息无法结束，所以可以设置成较长的循环或者设置成ctrl+C的信号结束

**5.3 生产者-消费者同步控制**

1）例如在生产者线程中，需要获取到当前为空的数据块的数量，就想到信号量SPACE的值与空数据块数量的值应是一致的，于是就用sem\_getvalue函数获取信号量值；但在经过多次测试后，发现这样会有一个问题，比如在消费者最后释放信号量MUTEX与SPACE时



在MUTEX释放而SPACE还没释放时，可能马上会有其他生产者访问临界资源，而此时SPACE的信号量值还没有进行加一，所以在生产者线程中会获取到错误的值导致出错

所以解决方法是新增两个变量来维护空数据块数量与已有数据块数量的值，而不是通过信号量来获取

2）本来是想的是生产者是顺序存储，消费者是顺序读取，但考虑到符合实际情况，改为了都是随机从某个数据块读/写

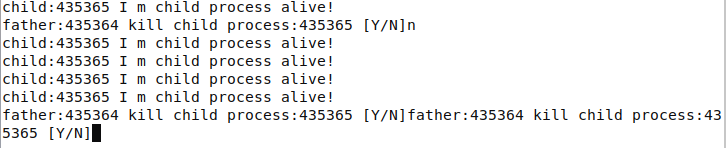
**5.4 进程通信**

1）当父进程打印信息时因为子进程依然在运行，所以两者打印的信息会混在一起



于是利用信号机制，在父进程要打印信息时用SIGSTOP暂停子进程，之后再用SIGCONT信号恢复

2）以及值得注意的是在父进程打印第一次信息后，以后每次打印并不是每两秒打印一次，而是每四秒一次性打印两遍



后来发现比如输入”n”时是按一下n再按回车，第二个回车符会被当作下一次的输入，所以用一个getchar吃掉回车符就行了

**六、实验参考资料和网址**

**（1）教学课件**

**（2）**[【Linux】Linux的共享内存\_Yngz\_Miao的博客-CSDN博客\_linux 共享内存](https://blog.csdn.net/qq_38410730/article/details/81488145)

**（3）[【Linux函数】Signal ()函数详细介绍\_华秋实的博客-CSDN博客\_linux signal函数是线程级的吗](https://blog.csdn.net/yockie/article/details/51729774)**

**（4）** [Linux 下的KILL函数的用法 - 拂 晓 - 博客园 (cnblogs.com)](https://www.cnblogs.com/leeming0222/articles/3994125.html)

**（5） [进程同步与互斥：LINUX环境\_虚神公子的博客-CSDN博客\_进程的同步与互斥linux](https://blog.csdn.net/m0_52435951/article/details/118253373)**

**……….**