

Operating System Principle, OS

《操作系统原理实验(2)》

Linux+ Windows综合版本

华中科技大学网安学院

2025年10月-2026年01月

实验二：第4章进程管理，第5章死锁

● 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/Windows下创建2个线程A和B，循环输出数据或字符串。
- 2) 在Linux下创建一对父子进程，实验wait同步函数。
- 3) 在Windows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printf调试进程创建和调度策略的相关信息。

● 三、实验要求

- 1,2,4,6必做,其余任选1。课前提前预做，老师机房检查和答疑。
- 现场检查：任意2个通过得2分。**21点开始检查，21:50检查结束。**

实验二：第4章进程管理，第5章死锁

● 四、实验指南

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

◆ 提示1：使用pthread线程库

◆ 提示2：线程A递增输出1-1000；线程B递减输出1000-1。为避免输出太快，每隔0.2秒（可自行调节）输出一个数。

◆ 提示3：输出数据时，同时输出A或B以标示是哪个线程输出的，并注意格式化输出信息。例如：

A:1000

A:0999

B:0001

A:0998

B:0002

.....

实验二：第4章进程管理，第5章死锁

● 四、实验指南

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

◆ 提示1：使用CreateThread创建线程

◆ 提示2：线程A递增输出1-1000；线程B递减输出1000-1。为避免输出太快，每隔0.2秒输出一个数。

◆ 提示3：输出数据时，同时输出A或B标示是哪个线程输出的，并注意格式化输出信息。例如：

A:1000

A:0999

B:0001

A:0998

B:0002

.....

■ 2) 在Linux下创建父子进程，实验wait同步函数，理解父子进程同步

◆ 提示1：子进程休眠5秒，父进程不休眠。子进程用exit返回参数。

◆ 提示2：父进程调用wait等待子进程先结束，并分析子进程返回参数。

◆ 提示3：父进程输出子进程的返回信息。

实验二：第4章进程管理，第5章死锁

● 四、实验指南

■ 3) 在Windows下，利用线程实现并发画圆画方。

◆ 提示1：圆心，半径，颜色，正方形中心，边长，颜色自己确定。

◆ 提示2：圆和正方形边界建议都取720个点。为直观展示绘制过程，每个点绘制后睡眠0.2秒~0.5秒。

◆ 提示3：建议使用VS和MFC或QT对话框类型程序来绘制窗口和图形。

实验二：第4章进程管理，第5章死锁

● 四、实验指南

■ 4) 在Windows或Linux下利用线程实现“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： Windows使用**临界区对象**和**信号量对象**，主要函数
EnterCriticalSection | LeaveCriticalSection | WaitForSingleObject |
ReleaseSemaphore
- ◆ 提示3： 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4： 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5： 消费者每休眠100ms-1s的随机时间消费一个数据。
- ◆ 提示6： 屏幕打印（或日志文件记录）每个数据的生产和消费记录。

实验二：第4章进程管理，第5章死锁

● 四、实验指南

■ 5) 在Linux下利用信号机制实现进程通信

- ◆ 提示1：父进程创建子进程，并让子进程进入死循环。
- ◆ 提示2：子进程每隔2秒输出”I am Child Process, alive !\n”
- ◆ 提示3：父进程询问用户”To terminate Child Process. Yes or No? \n”
要求用户从键盘回答Y或N.若用户回答N，延迟2秒后再提问.
- ◆ 提示4：若用户回答Y，向子进程发送**用户信号**，让子进程结束。
- ◆ 提示5：子进程结束之前打印字符串：“Bye,Wolrd !\n”
- ◆ 提示6：函数：kill(), signal(), 利用**用户信号**，编写**信号处理函数**

实验二：第4章进程管理，第5章死锁

● 四、实验指南

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

- ◆ 提示1：同时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。Windows尝试使用临界区对象（EnterCriticalSection, LeaveCriticalSection）；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。Windows尝试使用 WaitForMultipleObjects, WaitForSingleObject 和互斥量对象 ReleaseMutex 等相关函数) 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。

实验二：第4章进程管理，第5章死锁

- 7) 研读Linux内核并用printf调试进程创建和调度策略的相关信息。
 - 要求：编写应用程序Hello.c，调用fork创建进程，在内核中跟踪该新建子进程的fork过程和显示与调度策略相关的PCB成员变量。
 - 提示1：编写应用程序Hello.c，在其中调用fork创建子进程(功能不限)，打印出父子进程的ID号；
 - 提示2：在内核中合适的位置（譬如do_fork函数内的某处）用printf输出“当前正创建的进程对应cmd，进程ID和父进程ID”等调试信息。
 - 提示3：为避免do_fork函数频繁地输出上述调试信息，须限定仅在Hello程序中调用fork时才输出上述调试信息，请思考要如何实现。
 - 参考方法：内核设计全局变量bool flag和系统调用SetDebug(bool)，SetDebug可以修改flag的值为true或false。在Hello程序中的调用fork函数前后，分别调用SetDebug(true)和SetDebug(false)修改flag。在printf调试信息时检查flag以确定是否要使用printf输出调试信息

线程-互斥锁-信号量

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string>
4
5 #include <pthread.h>
6 #include <semaphore.h>
7
8 using namespace std;
9
10#define BUFFER_LENGTH 32 //队列长度
11int iBuffer[ BUFFER_LENGTH ];
12//互斥锁
13pthread_mutex_t mutexBuffer = PTHREAD_MUTEX_INITIALIZER;
14//信号量
15sem_t semFull; //信号量：数据的个数
16sem_t semEmpty; //信号量：空槽的个数
17
18int nBufferWritePos = 0; //队尾
19int nBufferReadPos = 0; //队头
20
```

线程-互斥锁-信号量

```
21 void* threadProducer(void* arg)
22 {
23     while(true)
24     {
25         sem_wait(&semEmpty); //若空槽个数低于0阻塞
26         pthread_mutex_lock(&mutexBuffer);
27
28         //准备数据
29         srand(time(0));
30         int iRand = rand() / 1000;
31
32         //写入缓冲区(写入位置: nBufferWritePos)
33         iBuffer[nBufferWritePos] = iRand;
34         //更新写入位置: nBufferWritePos
35         nBufferWritePos = (nBufferWritePos + 1) % BUFFER_LENGTH;
36
37         pthread_mutex_unlock(&mutexBuffer);
38         sem_post(&semFull);
39     }
40     return (void*)0;
41 }
```

线程-互斥锁-信号量

```
43 void* threadConsumer(void* TaskParam)
44 {
45     while(true)
46     {
47         //printf("consumer is preparing data\n");
48         sem_wait(&semFull); //若填充个数低于0阻塞
49         pthread_mutex_lock(&mutexBuffer);
50         //读出数据(位置: nBufferReadPos)
51         printf("iBuffer[%5d] = %d\n", nBufferReadPos,
52                iBuffer[nBufferReadPos] );
53         //更新读出位置: nBufferReadPos
54         nBufferReadPos = (nBufferReadPos+1) % BUFFER_LENGTH;
55
56         pthread_mutex_unlock(&mutexBuffer);
57         sem_post(&semEmpty);
58     }
59
60     return (void*)0;
61 }
```

```
63 int main(int argc, char *argv[])
64 {
65     sem_init(&semFull, 0, 0 );
66     sem_init(&semEmpty, 0, BUFFER_LENGTH );
67
68     pthread_t threadHandleProducer[2];
69     pthread_t threadHandleConsumer[3];
70
71     int nErrThread = 0;
72
73     nErrThread = pthread_create(&threadHandleProducer[0], NULL, threadProducer, NULL);
74     nErrThread = pthread_create(&threadHandleProducer[1], NULL, threadProducer, NULL);
75
76     nErrThread = pthread_create(&threadHandleConsumer[0], NULL, threadConsumer, NULL);
77     nErrThread = pthread_create(&threadHandleConsumer[1], NULL, threadConsumer, NULL);
78     nErrThread = pthread_create(&threadHandleConsumer[2], NULL, threadConsumer, NULL);
79
80     int nThreadNo = 0;
81
82     for( nThreadNo = 0;nThreadNo < 3; nThreadNo++ )
83     {
84         pthread_join(threadHandleConsumer[nThreadNo], NULL);
85     }
86
87     for( nThreadNo = 0;nThreadNo < 2; nThreadNo++ )
88     {
89         pthread_join(threadHandleProducer[nThreadNo], NULL);
90     }
91
92 }
```

线程-互斥锁-信号量

```
susg@ThinkPad:~/devroot/Studyc/OS_C$ g++ -o ThreadSemLock.out ThreadSemLock.cpp  
/tmp/cc7Bvgk3.o: In function `threadProducer(void*)':  
ThreadSemLock.cpp:(.text+0x12): undefined reference to `sem_wait'  
ThreadSemLock.cpp:(.text+0x92): undefined reference to `sem_post'  
/tmp/cc7Bvgk3.o: In function `threadConsumer(void*)':  
ThreadSemLock.cpp:(.text+0xad): undefined reference to `sem_wait'  
ThreadSemLock.cpp:(.text+0x111): undefined reference to `sem_post'  
/tmp/cc7Bvgk3.o: In function `main':  
ThreadSemLock.cpp:(.text+0x145): undefined reference to `sem_init'  
ThreadSemLock.cpp:(.text+0x159): undefined reference to `sem_init'  
ThreadSemLock.cpp:(.text+0x17b): undefined reference to `pthread_create'  
ThreadSemLock.cpp:(.text+0x19d): undefined reference to `pthread_create'  
ThreadSemLock.cpp:(.text+0x1bb): undefined reference to `pthread_create'  
ThreadSemLock.cpp:(.text+0x1dd): undefined reference to `pthread_create'  
ThreadSemLock.cpp:(.text+0x1ff): undefined reference to `pthread_create'  
ThreadSemLock.cpp:(.text+0x24e): undefined reference to `pthread_join'  
ThreadSemLock.cpp:(.text+0x278): undefined reference to `pthread_join'  
collect2: error: ld returned 1 exit status  
susg@ThinkPad:~/devroot/Studyc/OS_C$ g++ -o ThreadSemLock.out ThreadSemLock.cpp -lpthread  
susg@ThinkPad:~/devroot/Studyc/OS_C$ █
```

共享内存

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/shm.h>
4 #define SIZE 4
5 int main( )
6 {
7     int ShaMemID ;
8     int *ShaMemAddr ;
9     int pid ;
10    ShaMemID = shmget(IPC_PRIVATE, SIZE, IPC_CREAT|0600) ;
11    pid = fork( ) ;
12    if ( pid == 0 )
13    {
14        printf("CHild PID = %d.\n", getpid()) ;
15        ShaMemAddr = (int *) shmat( ShaMemID, NULL, 0 ) ;
16        (*ShaMemAddr) = getpid() ;
17        shmdt( ShaMemAddr ) ;
18        return 0;
19    }
20    else if ( pid > 0 )
21    {
22        sleep(3) ;
23        ShaMemAddr = (int *) shmat(ShaMemID, NULL, 0 ) ;
24        printf("CHild PID = %d (%ShaMemAddr).\n", *ShaMemAddr) ;
25        printf("Parent PID = %d\n", getpid()) ;
26        shmdt( ShaMemAddr ) ;
27        shmctl(ShaMemID, IPC_RMID, NULL) ;
28    }
29    return 0 ;
30 }
```

共享内存

```
susg@ThinkPad: ~/devroot/StudyC/OS_C
susg@ThinkPad:~/devroot/StudyC/OS_C$ g++ ShareMem.cpp -o ShareMem.out
susg@ThinkPad:~/devroot/StudyC/OS_C$ ./ShareMem.out
Child PID = 30731.
Child PID = 30731 (ShaMemAddr).
Parent PID = 30730
susg@ThinkPad:~/devroot/StudyC/OS_C$
```

信号机制：父进程发信号KILL子进程

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <signal.h>
5 #include <unistd.h>
6 #include <wait.h>
7
8 void handler(int arg)
9 {
10     printf("Bye, World!\n");
11     exit(0);
12 }
13
```

信号机制：父进程发信号KILL子进程

- 子进程把**PID**如何给父进程

- PS查看子进程**PID**, 父进程运行时输入子进程**PID**
- 子进程自打印出**PID**, 父进程运行时输入子进程**PID**
- 共享内存

```
14 int main(int argc, const char *argv[])
15 {
16
17     pid_t pid;
18     pid = fork();
19     if(pid == 0) //子进程
20     {
21         //注册信号回调函数，当信号发生会调用handler
22         signal(SIGUSR1, handler);
23         while(true)
24         {
25             printf("Child Process Alive! PID = %d.\n", getpid());
26             sleep(2);
27         }
28     }
```

信号机制：父进程发信号KILL子进程

```
8 void handler(int arg)
9 {
10     printf("Bye, World!\n");
11     exit(0);
12 }
13
14 int main(int argc, const char *argv[])
15 {
16
17     pid_t pid;
18     pid = fork();
19     if(pid == 0) //子进程
20     {
21         //注册信号回调函数，当信号发生会调用handler
22         signal(SIGUSR1, handler);
23         while(true)
24         {
25             printf("Child Process Alive! PID = %d.\n", getpid());
26             sleep(2);
27         }
28     }
```

信号机制：父进程发信号KILL子进程

```
28 //父进程
29 {
30     char input;
31     int ChildPID;
32
33     sleep(4);
34
35     printf("Please Input Child Process PID:");
36     scanf("%d", &ChildPID);
37
38     while(true)
39     {
40         sleep(5); //睡 2 秒
41         printf("To Terminate Child Process %d, Yes or No: \n", ChildPID);
42         scanf("%c", &input);
43         if((input == 'Y') || (input == 'y'))
44         {
45             kill(ChildPID, SIGUSR1);
46             wait(NULL); //等待回收子进程的资源
47             break;
48         }
49     }
50 }
51 return 0;
52 }
```