# 实验五 Posix信号量

1. 线程互斥例题：黑白子拣子问题

进程A专门拣黑子，进程B专门拣白子

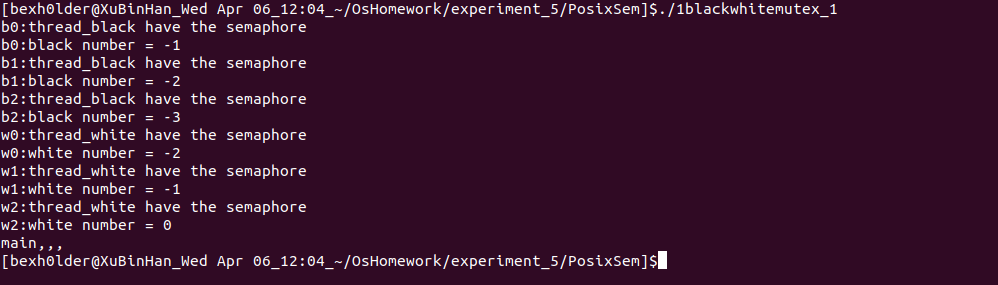
每个进程每次只能拣一个子，当一个进程拣子时不允许另一个进程去拣子

用PV操作解决互斥方法

* 1. 确定临界资源，找到临界去；
  2. 设置互斥信号量，初始值为1；
  3. 在临界区前P操作，临界区后V操作。

编译链接：gcc -o 1blackwhitemutex\_1 1blackwhitemutex\_1.c -lpthread

1. 了解信号量的初始化操作、P操作、V操作



初始化信号量为1使一个线程执行P操作运行并使信号量变为0，另一个线程执行 P操作后信号量变为-1并阻塞，运行的线程运行完执行V操作使信号量变为0之后 另一个进程运行，执行完主要操作后执行V操作使信号量变为1如此往复,达到互 斥效果

信号量初始化：sem\_init(sem\_t \*sem, int pshared, unsigned int value);

参数说明：

sem\_t \*sem：信号量变量

int pshared： 指明信号量的类型。不为0时此信号量在相关进程间共享，否则只 能为当前进程的所以线程共享

unsigned int value：该参数指定信号量的初始值

返回值：成功时返回0：错误时，返回-1，并将errno设置为合适的值

P操作（信号量的-1操作）：int sem\_wait(sem\_t \*sem);

参数说明：

sem\_t \*sem：信号量变量

函数说明：

等待信号量，如果信号量的值大于0，将信号量的值减1，立即返回。如果信号量 的值为0，则线程阻塞。

返回值：成功返回0，失败返回-1。

V操作（信号量的+1操作）：int sem\_post(sem\_t \*sem);

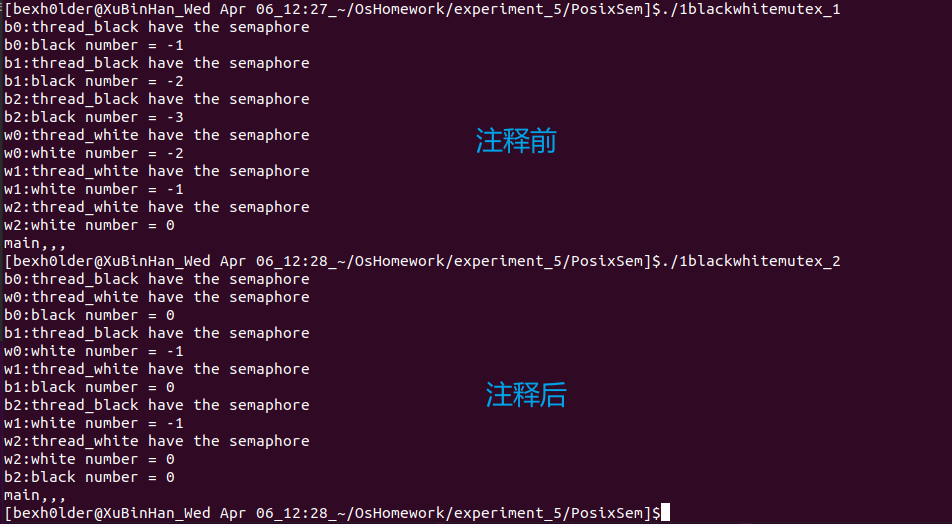
参数说明：

sem\_t \*sem：信号量变量

函数说明：

释放信号量，让信号量的值加1。若此时有sem\_wait正在阻塞则唤醒。

1. 注释红色代码，保留sleep语句，体会如何使用信号量对共享资源的互斥访问进行管理。



信号量初值为1，thread\_black\_fun线程执行P操作后信号量变为0， thread\_white\_fun线程执行P操作时信号量是0，是thread\_white\_fun线程阻塞无

法运行，从而实现互斥

1. 线程同步例题：文件读取问题

四个线程，两个Read线程从文件中读取数据，两个Handle线程处理取出来的数据，要 求先读取数据，后处理数据，两组线程是同步关系

用PV操作解决同步的方法：

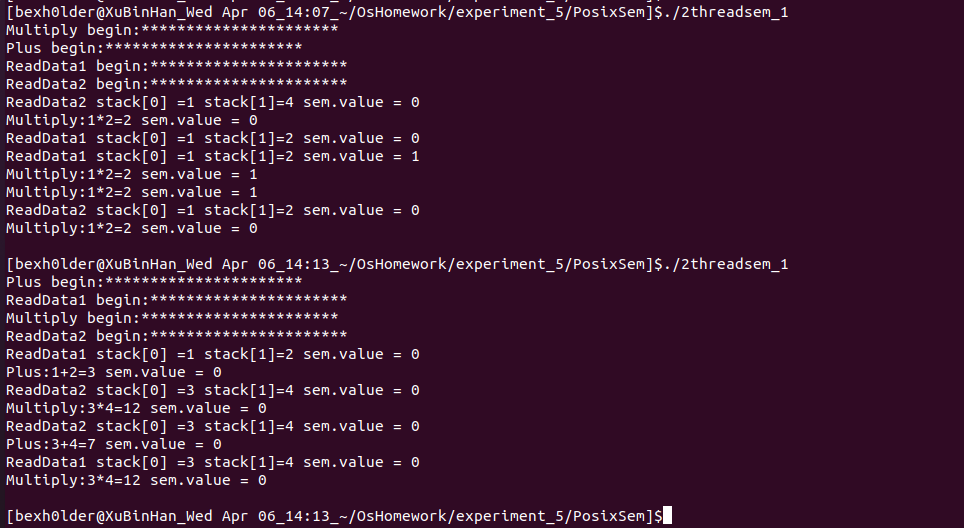
* 1. 确定先后动作；
  2. 设置同步信号量，初始值为0；
  3. 先动作后V操作，后动作前P操作

编译链接：

1. 理解信号量的初始化操作、P操作和V操作

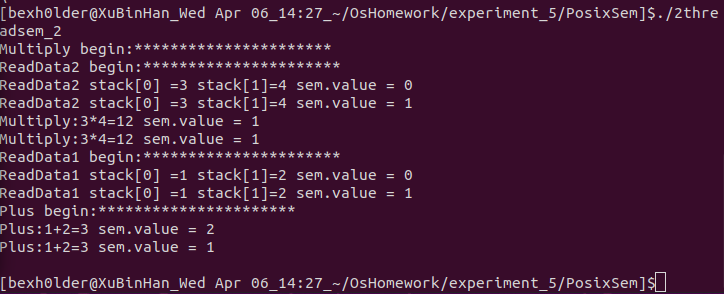
初始信号量为0，后动作的P操作执行后检测到信号量为-1，使后动作的线程阻塞，先动作线程主要操作执行完之后执行V操作，使信号量重归于0，后动作线程执行

1. 分析出四个线程的前驱后继关系



两个ReadData线程并发执行，初始信号量为0，而两个HandleData线程在运行主 要操作前都由P操作，所以只有执行过ReadData线程的V操作之后两个HandleData 才可运行。

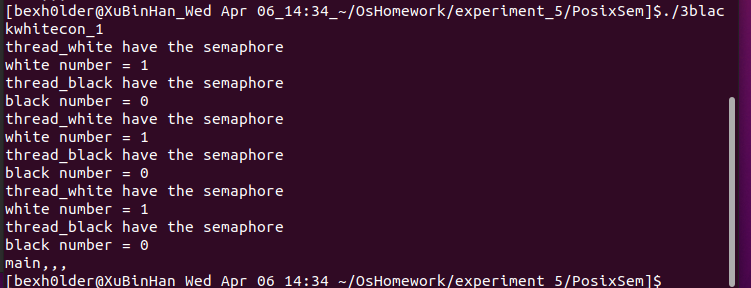
1. 若要求ReadData1读出的数据仅由HandleData1处理，ReadData2读出的数据仅由 HandleData2处理，请修改例程
2. #include <stdio.h>
3. #include <pthread.h>
4. #include <semaphore.h>
5. #define MAXSTACK 100
6. int stack[MAXSTACK];
7. int size=0;
8. sem\_t sem1;
9. sem\_t sem2;
10. */\* 从文件1.dat读取数据，每读一次，信号量加一\*/*
11. void ReadData1(void){
12. printf("ReadData1 begin:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
13. FILE \*fp=fopen("1.dat","r");
14. while(!feof(fp)){
15. fscanf(fp,"%d %d",&stack[0],&stack[1]);
16. sem\_getvalue(&sem1,&size);
17. printf("ReadData1 stack[0] =%d stack[1]=%d sem.value = %d\n",stack[0],stack[1],size);
18. sem\_post(&sem1);
19. *//sleep(1);*
20. }
21. fclose(fp);
22. }
23. */\*从文件2.dat读取数据\*/*
24. void ReadData2(void){
25. printf("ReadData2 begin:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
26. FILE \*fp=fopen("2.dat","r");
27. while(!feof(fp)){
28. fscanf(fp,"%d %d",&stack[0],&stack[1]);
29. sem\_getvalue(&sem2,&size);
30. printf("ReadData2 stack[0] =%d stack[1]=%d sem.value = %d\n",stack[0],stack[1],size);
31. sem\_post(&sem2);
32. *//sleep(1);*
33. }
34. fclose(fp);
35. }
36. */\*阻塞等待缓冲区有数据，读取数据后，释放空间，继续等待\*/*
37. void HandleData1(void){
38. printf("Plus begin:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
39. while(1){
40. sem\_getvalue(&sem1,&size);
41. sem\_wait(&sem1);
42. printf("Plus:%d+%d=%d sem.value = %d \n",stack[0],stack[1],stack[0]+stack[1],size);
43. *//sleep(1);*
44. }
45. }
46. void HandleData2(void){
47. printf("Multiply begin:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
48. while(1){
49. sem\_getvalue(&sem2,&size);
50. sem\_wait(&sem2);
51. printf("Multiply:%d\*%d=%d sem.value = %d\n",stack[0],stack[1],stack[0]\*stack[1],size);
52. *//sleep(1);*
53. }
54. }
55. int main(void){
56. pthread\_t t1,t2,t3,t4;
57. sem\_init(&sem1,0,0);
58. sem\_init(&sem2,0,0);
59. pthread\_create(&t1,NULL,(void \*)HandleData1,NULL);
60. pthread\_create(&t2,NULL,(void \*)HandleData2,NULL);
61. pthread\_create(&t3,NULL,(void \*)ReadData1,NULL);
62. pthread\_create(&t4,NULL,(void \*)ReadData2,NULL);
63. */\* 防止程序过早退出，让它在此无限期等待\*/*
64. getchar();
65. }



1. 例程1中，如果还要求例程A和例程B交替拣子，则为进程同步问题，修改代码并调试 运行

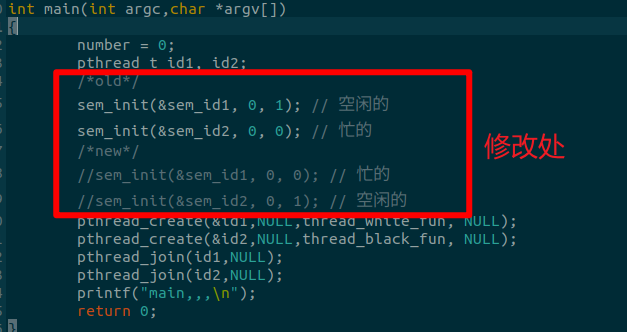
编译链接

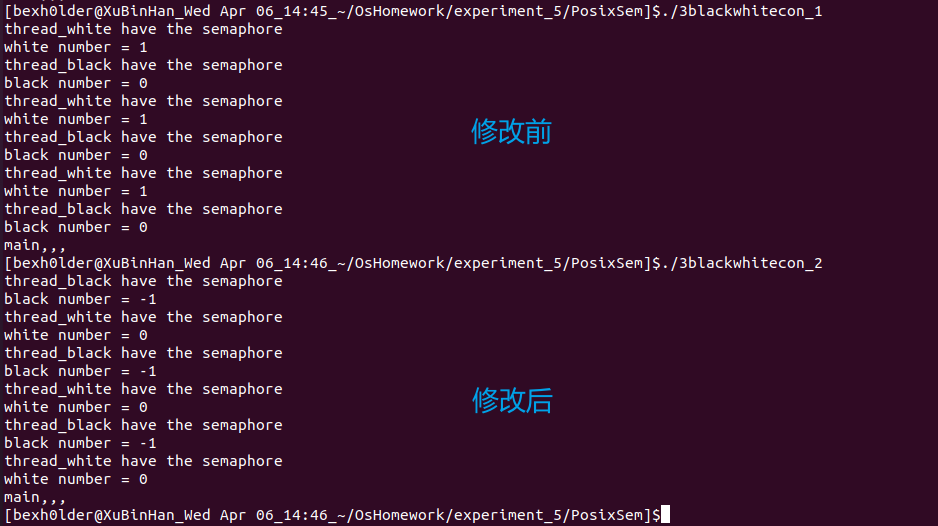
1. 理解信号量的初始化操作、P操作和V操作



初始化两个信号量id1和id2，id1为1，id2为0，thread\_white\_fun线程包含了对 id1的P操作和对id2的V操作，thread\_black\_fun线程包含了对 id2的P操作和对 id1的V操作，因为初始id1为1，id2为0，所以thread\_white\_fun线程运行， thread\_black\_fun线程阻塞，thread\_white\_fun线程运行完之后，id1为0，id2为 1，使thread\_black\_fun线程运行，thread\_black\_fun线程运行完之后，id1为1，id2 为0，如此往复实现交替拣子

1. 修改信号量的初始值，观察线程同步的顺序变化





1. 用Posix信号解决“写者优先的读者-写者问题”
2. 理解信号量的初始化操作，P操作和V操作



初始化x信号量和wsem的信号量均为1，读写进程不能同时进行，wsem信号量P V操作实现读写操作互斥，保证只有一个写进程进行或由一个或多个读进程进行， 而x 的PV操作控制读者的增加，读者无法同时增加减少

1. 观察线程并发，理解读者优先

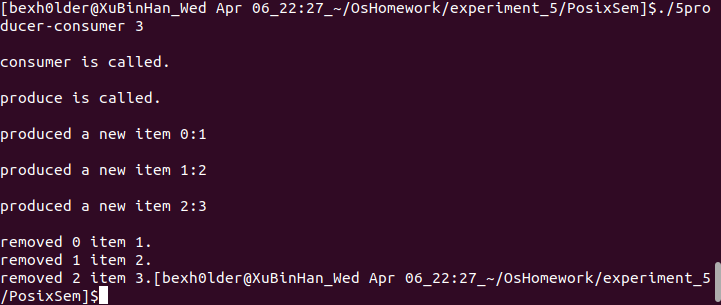
读进程具有优先权，一个读进程开始访问数据区时，至少有一个读进程在读，就为 读进程保留临界资源的控制权，而读进程只有全部完成，写进程才可执行，因此写 进程有可能处于饥饿状态

1. 用POSIX信号量解决生产者/消费者问题

使用信号量解决有限缓冲区生产者/消费者问题的方法如下：

编译链接

1. 理解信号量的初始化操作、P操作和V操作



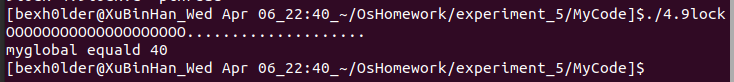
mutex初始值为1使操作能够进行，mutex的PV操作实现生产者和消费者之间的 互斥，full的PV操作控制消费者的执行，empty的PV操作控制生产者的执行

1. 观察线程并发，理解生产者消费者在信号量的控制下，实现供需平衡

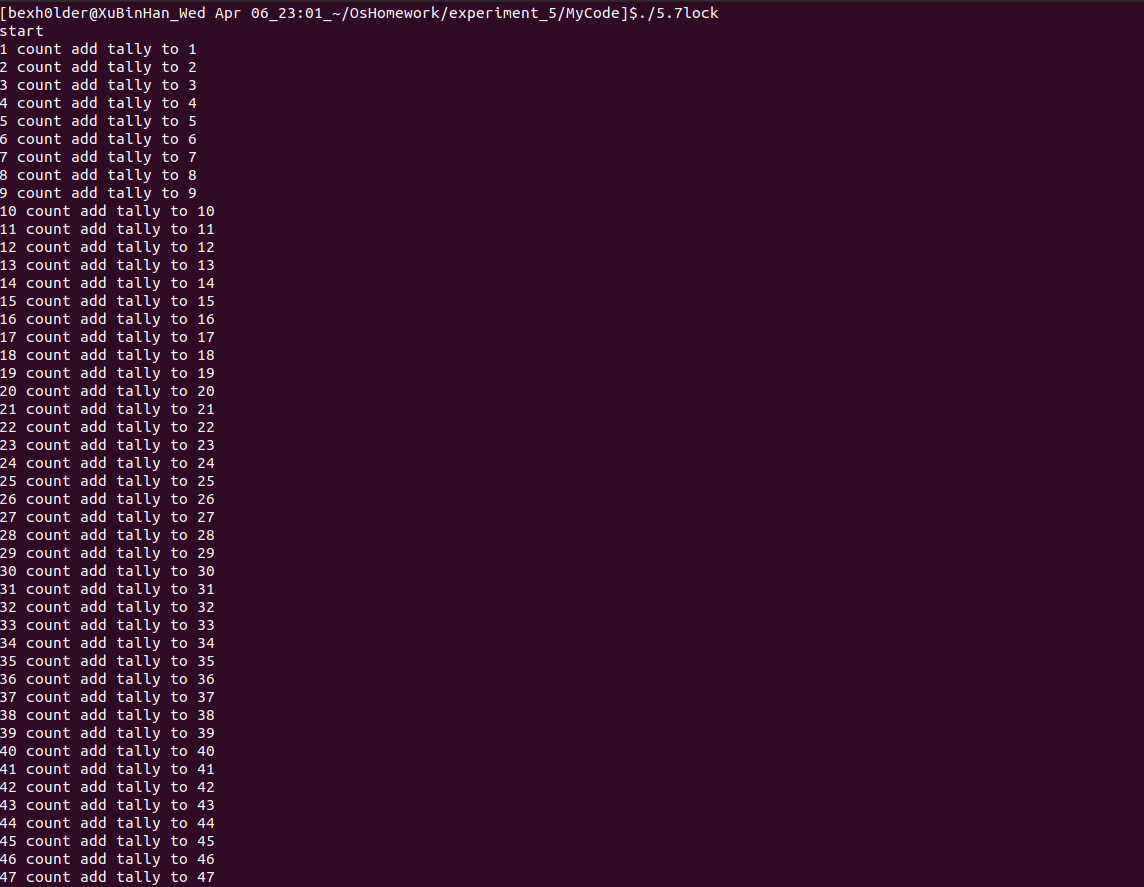
初始状态full为0，empty为槽数，所以生产者线程先执行，mutex控制同一情况 下只可有一个线程执行，当full为0时只能是生产者执行，当empty为0时只能是 消费者执行，当full和empty都不为0时，线程并发执行生产者和消费者相互争夺 执行

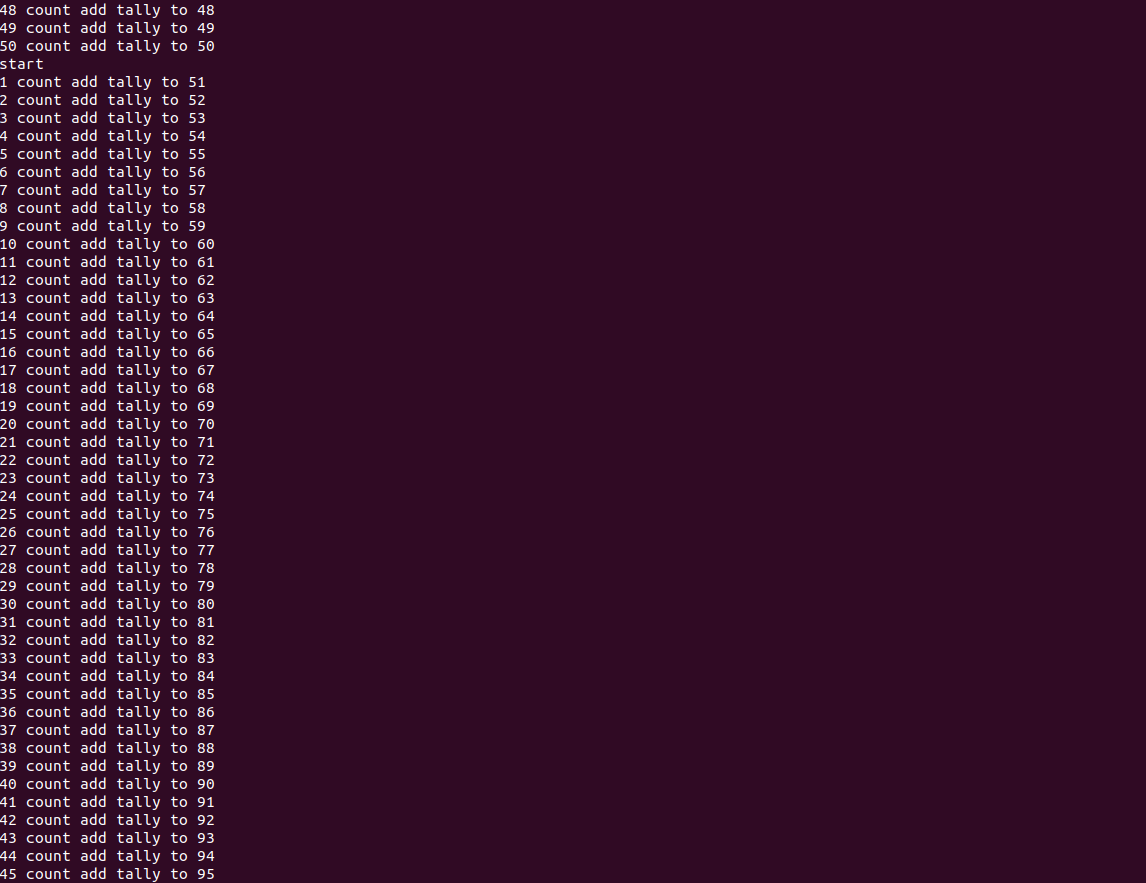
编程题：

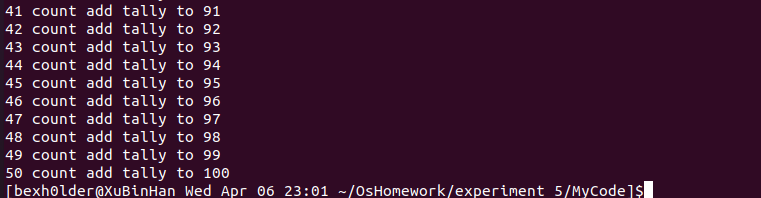
1. 课后习题4.9
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <pthread.h>
5. #include <errno.h>
6. #include <unistd.h>
7. */\*互斥量 \*/*
8. pthread\_mutex\_t mutex;
9. int myglobal=0;
10. void \*thread\_function(void \*arg)
11. {
12. int i,j;
13. for(i = 0; i <20; i++){
14. pthread\_mutex\_lock(&mutex); */\*获取互斥锁\*/*
15. j = myglobal;
16. j++;
17. printf(".");
18. fflush(stdout);
19. usleep(1000);
20. myglobal = j;
21. pthread\_mutex\_unlock(&mutex); */\*释放互斥锁\*/*
22. }
23. return NULL;
24. }
25. int main(){
26. pthread\_t mythread;
27. int i;
28. */\*互斥初始化\*/*
29. pthread\_mutex\_init (&mutex, NULL);
30. if( pthread\_create(&mythread,NULL,thread\_function,NULL))
31. {
32. printf("error creating thread!\n");
33. abort();
34. }
35. for(i = 0; i <20; i++)
36. {
37. pthread\_mutex\_lock(&mutex); */\*获取互斥锁\*/*
38. myglobal++;
39. printf("O");
40. fflush(stdout);
41. usleep(1000);
42. pthread\_mutex\_unlock(&mutex); */\*释放互斥锁\*/*
43. }
44. if( pthread\_join(mythread,NULL))
45. {
46. printf("error joining thread!\n");
47. abort();
48. }
49. printf("\nmyglobal equald %d\n",myglobal);
50. exit(0);
51. }



1. 课后习题5.7
2. #include <stdio.h>
3. #include <semaphore.h>
4. #include <pthread.h>
5. const int n = 50;
6. int tally;
7. sem\_t s;
8. void total()
9. {
10. printf("start\n");
11. int count;
12. for(count = 1; count <= n; count++){
13. sem\_wait(&s);
14. tally++;
15. printf("%d count add tally to %d\n",count,tally);
16. sem\_post(&s);
17. }
18. }
19. void main()
20. {
21. pthread\_t t1,t2;
22. tally = 0;
23. sem\_init(&s,0,1);
24. pthread\_create(&t1,NULL,(void\*)total,NULL);
25. pthread\_create(&t2,NULL,(void\*)total,NULL);
26. pthread\_join(t1,NULL);
27. pthread\_join(t2,NULL);
28. }







1. 生产者-消费者例程变形
2. #include <stdlib.h>
3. #include <pthread.h>
4. #include <stdio.h>
5. #include <unistd.h>
6. #include <semaphore.h>
7. #include <errno.h>
8. #include <fcntl.h>
9. char information;
10. int messages;  *//the number of messages*
11. sem\_t mutex,B\_empty,C\_empty,D\_empty,B\_full,C\_full,D\_full;
12. void \*send()
13. {
14. int i;
15. printf("\nsend is called.\n");
16. for(i=0;i<messages;i++)
17. {
18. sem\_wait(&B\_empty);
19. sem\_wait(&C\_empty);
20. sem\_wait(&D\_empty);
21. sem\_wait(&mutex);  *//锁住槽位，对于多个生产者的时候有必要，单个生产者没有必要*
22. printf("\nplz input the message information\n");
23. scanf("%c",&information);
24. getchar(); *//Avoid carriage return from being typed*
25. printf("\nA sends a message %c\n",information);
26. sleep(1);
27. sem\_post(&mutex);  *//释放锁*
28. sem\_post(&B\_full);
29. sem\_post(&C\_full);
30. sem\_post(&D\_full);
31. }
32. return NULL;
33. }
34. void \*acceptB()
35. {
36. int i;
37. printf("\nacceptB is called.\n");
38. for(i=0;i<messages;i++)
39. {
40. sem\_wait(&B\_full);  *//判断缓冲区中是否有条目，有的话将条目数减少1*
41. sem\_wait(&mutex);  *//锁住缓冲区，对多个消费者有必要，对单个消费者没必要*
42. printf("\nB accepts a message %c\n",information);
43. sleep(1);
44. sem\_post(&mutex);  *//释放锁*
45. sem\_post(&B\_empty); *//将缓冲区中的空槽数目加1*
46. }
47. return NULL;
48. }
49. void \*acceptC()
50. {
51. int i;
52. printf("\nacceptC is called.\n");
53. for(i=0;i<messages;i++)
54. {
55. sem\_wait(&C\_full);  *//判断缓冲区中是否有条目，有的话将条目数减少1*
56. sem\_wait(&mutex);  *//锁住缓冲区，对多个消费者有必要，对单个消费者没必要*
57. printf("\nC accepts a message %c\n",information);
58. sleep(1);
59. sem\_post(&mutex);  *//释放锁*
60. sem\_post(&C\_empty); *//将缓冲区中的空槽数目加1*
61. }
62. return NULL;
63. }
64. void \*acceptD()
65. {
66. int i;
67. printf("\nacceptD is called.\n");
68. for(i=0;i<messages;i++)
69. {
70. sem\_wait(&D\_full);  *//判断缓冲区中是否有条目，有的话将条目数减少1*
71. sem\_wait(&mutex);  *//锁住缓冲区，对多个消费者有必要，对单个消费者没必要*
72. printf("\nD accepts a message %c\n",information);
73. sleep(1);
74. sem\_post(&mutex);  *//释放锁*
75. sem\_post(&D\_empty); *//将缓冲区中的空槽数目加1*
76. }
77. return NULL;
78. }
79. int main(int argc,char \*argv[])
80. {
81. if(argc != 2)
82. {
83. printf("usage: prodcons <#itmes>\n");
84. exit(0);
85. }
86. pthread\_t t\_sendA,t\_acceptB,t\_acceptC,t\_acceptD;
87. messages = atoi(argv[1]);
88. *//创建信号量*
89. sem\_init(&mutex,0,1);
90. sem\_init(&B\_empty,0,1);
91. sem\_init(&C\_empty,0,1);
92. sem\_init(&D\_empty,0,1);
93. sem\_init(&B\_full,0,0);
94. sem\_init(&C\_full,0,0);
95. sem\_init(&D\_full,0,0);
96. pthread\_create(&t\_sendA,NULL,send,NULL);
97. pthread\_create(&t\_acceptB,NULL,acceptB,NULL);
98. pthread\_create(&t\_acceptC,NULL,acceptC,NULL);
99. pthread\_create(&t\_acceptD,NULL,acceptD,NULL);
100. pthread\_join(t\_sendA,NULL);
101. pthread\_join(t\_acceptB,NULL);
102. pthread\_join(t\_acceptC,NULL);
103. pthread\_join(t\_acceptD,NULL);
104. exit(0);
105. }

