**《操作系统》课第08次实验报告**

|  |  |
| --- | --- |
| 学院: | 软件学院 |
| 姓名: | 张怡桢 |
| 学号: | 2013747 |
| 邮箱: | 2662765987@qq.com |
| 时间: | 11/6/2022 |

1. **开篇感言**

“你长大后想成为什么人？”

“什么意思？长大后我就不能成为我自己了吗？”

-- 《阿甘正传》

2. **实验题目**

实现 I 个生产者 J 个消费者问题，其中共享缓冲区的大小为 N，所有生产者 共生产 K(K>N)个产品后结束，所有消费者共消费 K 个产品后结束。

3. **实验要求**

1. 严格按时序输出每个生产者、消费者的行为，其中包括生产产品 k、消费 产品 k、进入临界区、存入产品、取出产品、离开临界区；
2. 需要考虑边界（某生产者生产第 K 个产品后所有生产者结束；某消费者 消费第 K 个产品后所有消费者结束）
3. 需要考虑随机函数，生产者生产时需要一个随机时间；消费者消费时也 需要一个随机时间；
4. 编号：无论生产者还是消费者都需要有编号；产品同样也需要编号；缓 冲区的各个产品项也需要有编号；
5. 输出形式可以采用标准输出、图形动态显示及同时文字记录输出等方式，

无论是生产者还是消费者，其主要输出内容如下：

1. 进入临界区前，输出某某编号（生产者/消费者）线程准备进入临界 区
2. 进入临界区后，输出某某编号（生产者/消费者）线程已进入临界区
3. 离开临界区后，输出某某编号（生产者/消费者）线程已离开临界区
4. 生产者生产一个产品时，需要输出产品信息；
5. 生产者将产品放入缓冲区时，需要输出相关信息；
6. 消费者将产品从缓冲区取出时，需要输出相关信息；
7. 消费者消费一个产品时，需要输出产品信息；
8. \*不能出现竞态
9. \*\*不能出现忙等待

4. **原理方法**

1. 线程技术 CreateThread 或 \_beginthreadex
2. CreateThread
3. GetCurrentThreadId
4. GetThreadPriority
5. SetThreadPriority
6. Sleep
7. SuspendThread
8. ResumeThread
9. ExitThread
10. 线程同步
11. WaitForSingleObject
12. WaitForMultipleObjects
13. 随机数
14. srand ((DWORD)time(NULL)); /\* Seed the random # generator \*/
15. rand() /\* generate a random \*/
16. Sleep() /\*msec\*/
17. 临界区 CRITICAL\_SECTION
18. InitializeCriticalSection
19. DeleteCriticalSection
20. EnterCriticalSection
21. LeaveCriticalSection
22. 事件 event
23. CreateEvent
24. SetEvent
25. ResetEvent
26. PulseEvent
27. 互斥对象
28. CreateMutex
29. （OpenMutex）
30. WaitForSingleObject(hMutex, INFINITE)
31. ReleaseMutex
32. 信号量对象
33. CreateSemaphore
34. WaitForSingleObject(hSem, INFINITE)
35. ReleaseSemaphore

5. **代码分析**

5.1 **产品类的设计**

产品有相应的属性，自己的编号，生产者与生产的时间，存储的临界区的编号以及消费者与消费时间。

产品类拥有printProduct函数可以打印产品的信息。

|  |
| --- |
| C++ //产品类 class product {  public:  int id; //产品编号  int producerID; //生产者编号  char \*produceTime; //生产时间  char \*consumeTime; //生产时间  int spaceID; //缓冲区中存储编号  int consumerID; //消费者编号  //打印产品信息  void printProduct()  {   }  //构造函数  product(int id, int producerID, char \*produceTime, int spaceID)  {  this->id = id;  this->producerID = producerID;  this->produceTime = produceTime;  this->spaceID = spaceID;  this->consumerID = -1;  this->consumeTime = "-1";  }    }; |

5.2 **全局变量的设计：**

|  |
| --- |
| C++  // i生产者 int producer\_num; // j消费者 int consumer\_num; // k产品 int product\_num; // n缓冲区 int space\_num;  vector<product> space; //缓冲区 int produce\_num = 0; //生产的产品数量 int consume\_num = 0; //消耗的产品数量   cout << "请输入生产者数量："; cin >> producer\_num; cout << "请输入消费者数量："; cin >> consumer\_num; cout << "请输入产品数量："; cin >> product\_num; cout << "请输入缓冲区数量："; cin >> space\_num; if(producer\_num<=0){  cout<<"没有生产者，无法生产产品"<<endl;  return 0; } if(consumer\_num<=0){  cout<<"没有消费者，无法消费产品"<<endl;  return 0; } if(product\_num<=0){  cout<<"所需要产品为0，无需进行生产或消费"<<endl;  return 0; } if(space\_num<=0){  cout<<"缓存区为0,无法进行产品存储或消费"<<endl;  return 0; } |

5.3 **信号量设计**

本实验采用3个信号量实现线程间通信：

1）canProduce（表示生产者能够生产，初始值为3）：在canProduce不为0即缓冲区未满时，生产者生产，将canProduce减1；消费者消费一件产品后，缓冲区能再多放一件产品，canProduce加1。

2）canConsume（表示消费者能够消费，初始值为0）：在canConsume不为0即缓冲区中还有产品时，消费者消费，将canConsume减1；生产者生产一件产品后，缓冲区多放了一件产品，canConsume加1。

3）canSpace（表示能操作缓冲区，初始值为1）：在canSpace为1时对缓冲区进行操作，将其减1，操作完成后加1；为0时不能。实际上也能通过互斥量实现canSpace的功能。

|  |
| --- |
| C++ //信号量 sem\_t canProduce, canConsume, canSpace; sem\_init(&canProduce, 0, space\_num);//信号量：生产者可以存放产品的缓存数 sem\_init(&canConsume, 0, 0);//信号量：消费者可以消费的产品数 sem\_init(&canSpace, 0, 1);//信号量：是否可以访问缓存区 //回收信号量 sem\_destroy(&canProduce); sem\_destroy(&canConsume); sem\_destroy(&canSpace); |

5.4  **线程的设计**

|  |
| --- |
| C++ pthread\_t pid[producer\_num];//生产者线程 pthread\_t cid[consumer\_num];//消费者线程 //创建多线程  //创建生产者线程  for (int i = 0; i < producer\_num; i++)  {  pthread\_create(&pid[i], NULL, producer, (void \*)(i+1));  }  //创建消费者线程  for (int i = 0; i < consumer\_num; i++)  {  pthread\_create(&cid[i], NULL, consumer, (void \*)(i+1));  }  //回收线程  for (int i = 0; i < producer\_num; i++)  {  pthread\_join(pid[i], NULL);  }  for (int i = 0; i < consumer\_num; i++)  {  pthread\_join(cid[i], NULL);  } |

5.5 **生产者运行函数**

传入该线程的序号为参数，并且使用canProduce信号量以及canSpace信号量来控制生产者线程获得生产产品权以及进入临界区。

为加快程序执行，将sleep函数（秒）换成了usleep（微妙）

|  |
| --- |
| C++ //生产者 void \*producer(void \*param) {  long rank = (long)param;//生产者线程序号  while (true)  {  //如果生产和消费的产品数都超过product\_num，则退出线程  if (produce\_num >= product\_num && consume\_num >= product\_num)  exit(0);  //如果已生产的产品超过了product\_num，则退出循环  if (produce\_num >= product\_num)  break;  sem\_wait(&canProduce); //等待能够生产   sem\_wait(&canSpace); //等待能够操作缓冲区   //再次判断，防止有生产者在等待进入缓冲区的过程中所有生产者已经共生产了k件产品  if (produce\_num >= product\_num)  {   sem\_post(&canSpace);//释放缓冲区操作权  break;//退出循环  }  //生产耗时  int tmp = rand() % 5 + 1;   usleep(tmp);//生产耗时，生产线程进入睡眠  printSpace(1); //打印生产前的缓冲区状态  produce\_num++;//生产的产品数量加1  time\_t timep = time(0);  char \*produce\_time = ctime(&timep);//获取当前时间  produce\_time[strlen(produce\_time) - 1] = 0;//去掉换行符  product p = product(produce\_num, rank, produce\_time,(space.size()+1));//创建产品，记录产品id,producerID,produceTime,spaceID记  space.push\_back(p); //将生产的产品放入缓冲区  cout << "----------------------------------------------------------------------------------------------" << endl;    p.printProduct();   cout << "----------------------------------------------------------------------------------------------" << endl;  printSpace(2); //打印生产后的缓冲区状态  sem\_post(&canSpace); //释放缓冲区操作权  sem\_post(&canConsume); //消费者可以消费的信号量加一  }  return NULL; } |

5.6 **消费者运行的函数**

传入该线程的序号为参数，并且使用canConsume信号量以及canSpace信号量来控制消费者线程获得消费产品权以及进入临界区。

为加快程序执行，将sleep函数（秒）换成了usleep（微妙）

|  |
| --- |
| C++ //消费者 void \*consumer(void \*param) {  long rank = (long)param;//消费者线程序号  while (true)  {  //如果生产和消费的产品数都超过15，则退出  if (produce\_num >= product\_num && consume\_num >= product\_num)  exit(0);  //如果已消费的产品超过了product\_num，则退出循环  if (consume\_num >= product\_num)  break;  sem\_wait(&canConsume); //等待能够消费   sem\_wait(&canSpace);//等待能够操作缓冲区   //再次判断，防止有消费者等待进入缓冲区的过程中所有消费者已经共消费了prodect\_num件产品  if (consume\_num >= product\_num)  {   sem\_post(&canSpace);//释放缓冲区操作权  break;  }  //消费耗时  int tmp = rand() % 5 + 1;   usleep(tmp);//消费耗时，消费线程进入睡眠  printSpace(3); //打印消费前的缓冲区状态  consume\_num++;//消费的产品数量加1  time\_t timep = time(0);//获取当前时间  char \*consume\_time = ctime(&timep);//获取当前时间  consume\_time[strlen(consume\_time) - 1] = 0;//去掉换行符  space[space.size() - 1].consumerID = rank;//记录消费者ID  space[space.size() - 1].consumeTime = consume\_time;//记录消费时间  cout << "-------------------------------------------------------------------------------------------------" << endl;    space[space.size() - 1].printProduct();   cout << "--------------------------------------------------------------------------------------------------" << endl;  space.pop\_back(); //每次消费缓冲区vector的最后一件产品  printSpace(4); //打印消费后的缓冲区状态  sem\_post(&canSpace); //释放缓冲区操作权  sem\_post(&canProduce); //生产者可以生产的信号量加一  }  return NULL; } |

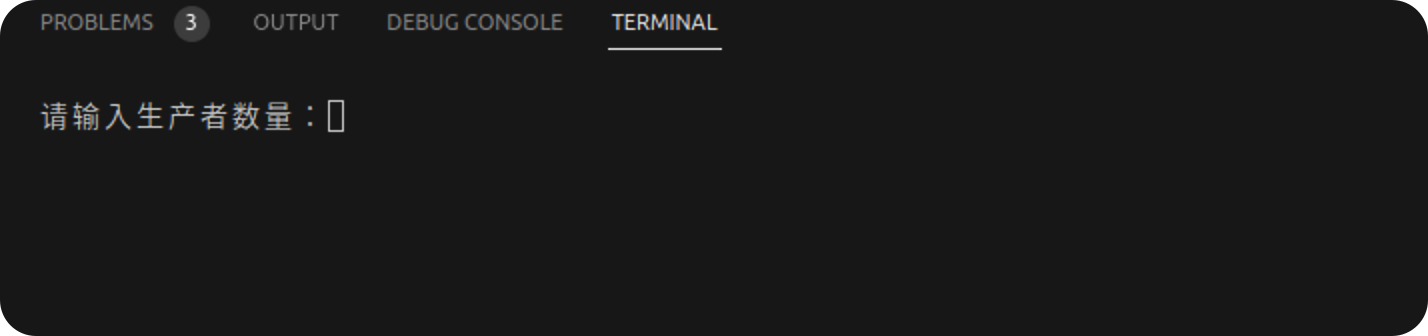
5.7 **打印缓冲区状态函数**

|  |
| --- |
| C++ //打印缓冲区状态 void printSpace(int state) {  if (state == 1)  cout << "【==========在生产前，";  else if (state == 2)  cout << "【==========在生产后，";  else if (state == 3)  cout << "【==========在消费前，";  else if (state == 4)  cout << "【==========在消费后，";  cout << "各缓冲区产品的信息如下==========】" << endl;  for (int i = 0; i < space\_num; i++)  {  if (space.size() < i + 1)  {   }  else  {   space[i].printProduct();  }  }  cout<<"【==================================================】"<<endl; } |

6. **具体步骤**

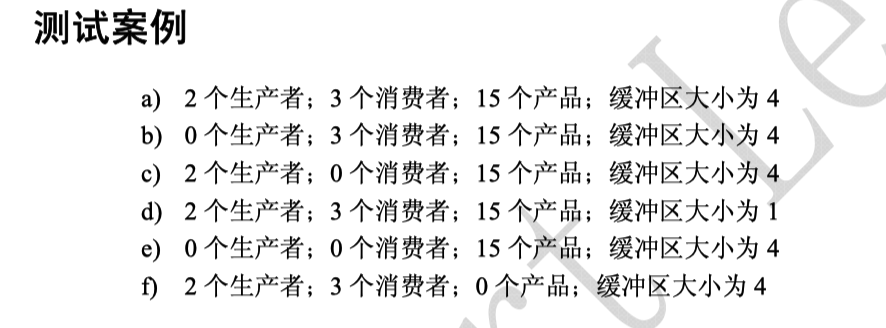
6.1 **编译并执行程序**

使用vscode进行编译并运行上述代码，代码可以正确运行

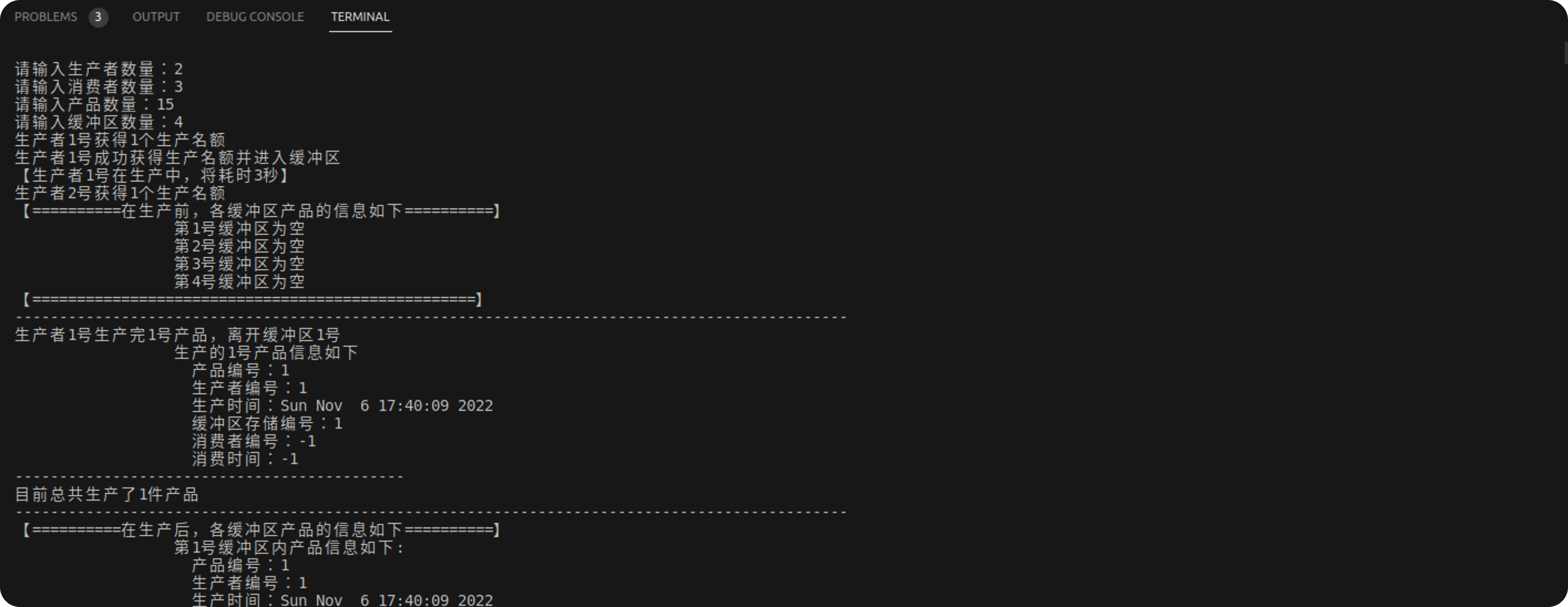


6.2 **测试**

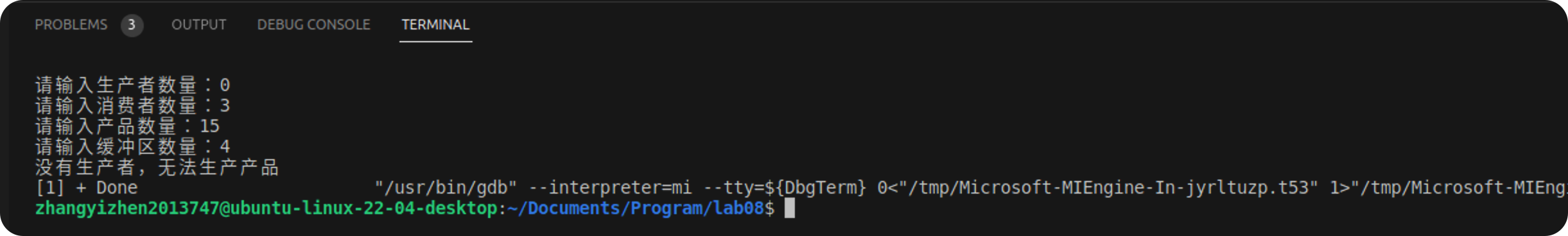
使用老师提供的pdf的测试案例进行测试，完整的结果在附件中，以txt文件保存输出的全部信息。



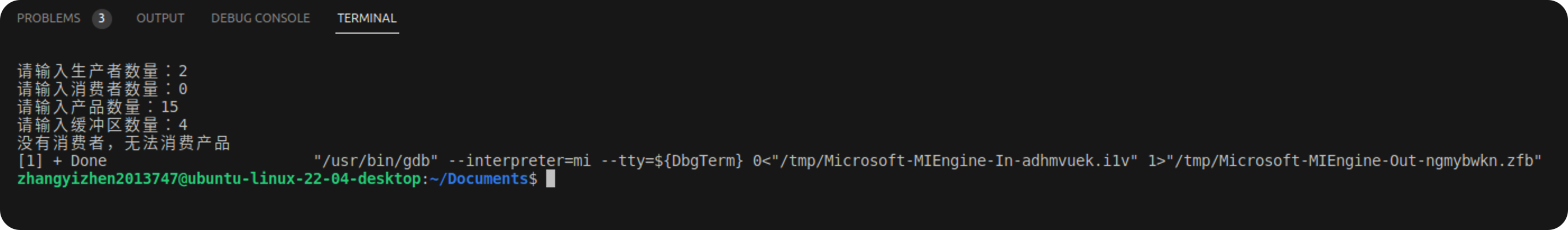
**a) 2 个生产者；3 个消费者；15 个产品；缓冲区大小为 4**



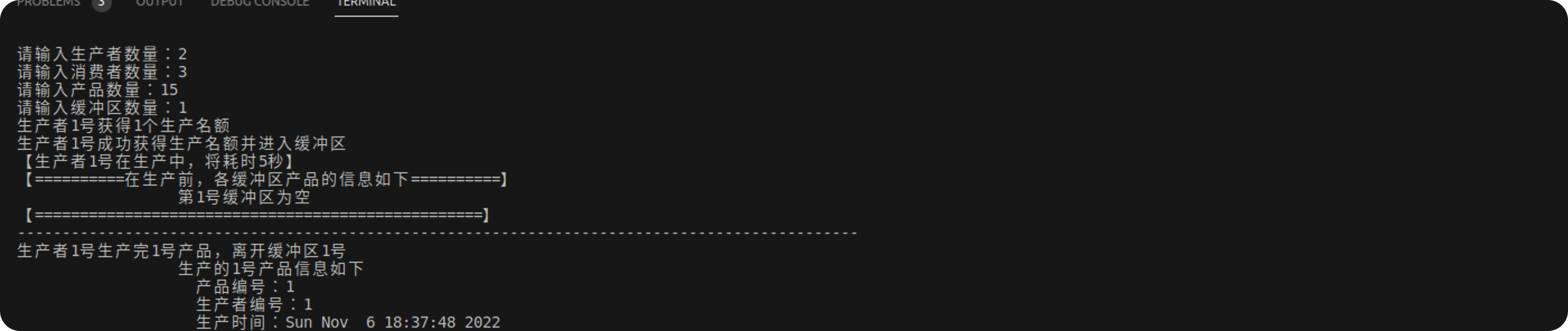
**b) 0 个生产者；3 个消费者；15 个产品；缓冲区大小为 4**



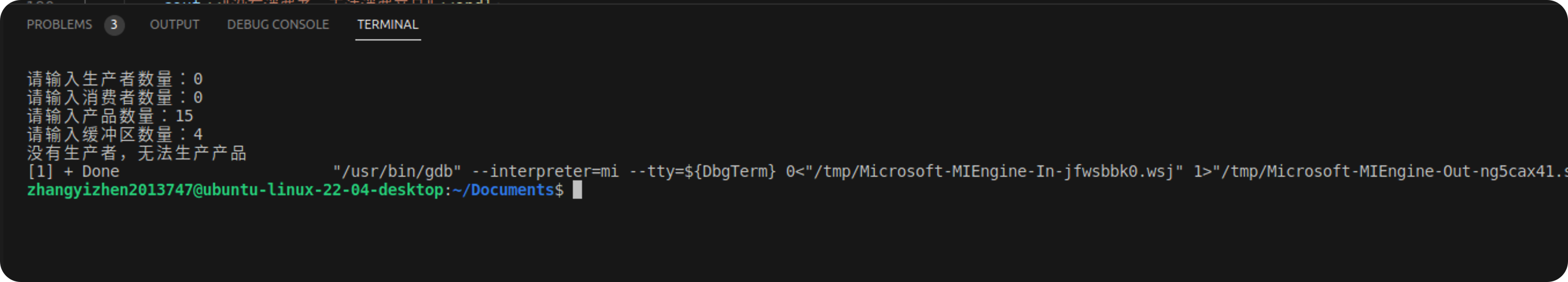
**c) 2 个生产者；0 个消费者；15 个产品；缓冲区大小为 4**



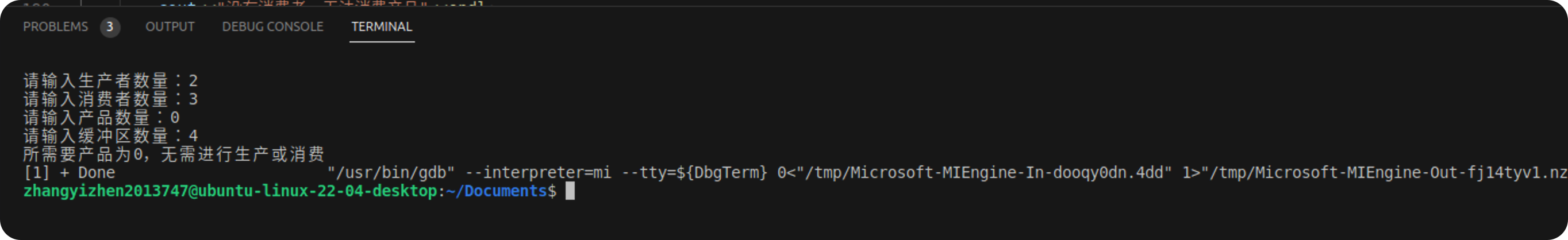
**d) 2 个生产者；3 个消费者；15 个产品；缓冲区大小为 1**



**e) 0 个生产者；0 个消费者；15 个产品；缓冲区大小为 4**



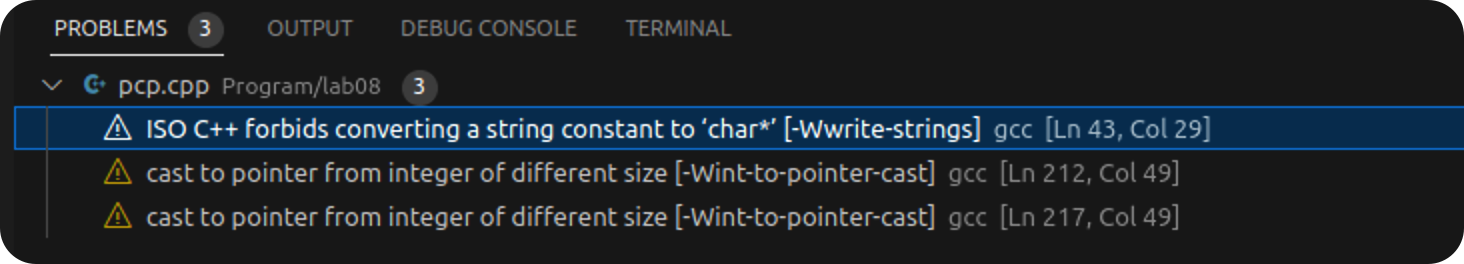
**f) 2 个生产者；3 个消费者；0 个产品；缓冲区大小为 4**



7. **心得体会**

经过对于生产者消费者模型的实现，以及对于信号量的设计与实现从而完成对于临界资源的控制访问，我对于多线程的理解更加深刻了。

不能将线程里两个wait的顺序调换，否则会出现死锁。例如将consumer的两个wait调换，在producer发出post信号后，如果producer线程此时再次获得运行机会，执行完了wait(canProduce)，此时，另一个consumer线程又获得运行机会，执行了wait(canSpace) ，如果此时缓冲区为空，那么consumer将会阻塞在wait(canConsume)，而producer也会因为无法缓冲区的操作权而阻塞在wait(canSpace)，这样两个线程都在阻塞，也就造成了死锁。



在编译的过程中，会报有三个警告，经验证，对于结果以及对于程序整体没有影响。

8. **参考资料**

[老师的github实验文档](https://github.com/albertleecn/osplab)

9. **附件**

详见其他案例的运行完整输出以及完整代码。