****

计算机操作系统实验2

生产者—消费者问题

姓名： 涂煜洋

学号： 71118119

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering

College of Software Engineering

Southeast University

2020年5月

1. **实验目的及内容**
2. **实验目的:**

通过实验，掌握Windows和Linux环境下互斥锁和信号量的实现方法，加 深对临界区问题和进程同步机制的理解，同时熟悉利用和进行多线程编程的方法。

1. **实验内容:**

1. 在Windows 操作系统上，利用Win32 API 提供的信号量机制，编写应用程序实现生产者——消费者问题。

2. 在 Linux操作系统上，利用提供的信号量机制，编写应用程序实现生产者——消费者问题。

3. 两种环境下，生产者和消费者均作为独立线程，并通过 empty、full、mutex 三个信号量实现对缓冲进行插入与删除。

4. 通过打印缓冲区中的内容至屏幕，来验证应用程序的正确性。

1. **实验基本知识与步骤**
2. **生产者—消费者问题的解决：**

生产者--消费者问题本质上就是有限缓冲的同步问题。解决这个同步问题需要3个信号量：，其中用于对缓冲池访问的互斥要求；用于表示缓冲区中被填充的项数；用于表示缓冲区中还未被填充的项数。

生产者为消费者生产满缓冲项，消费者为生产者生产空缓冲项，这使得我们将生产者与消费者看待为对称的两种对象，便可基本的伪代码结构：

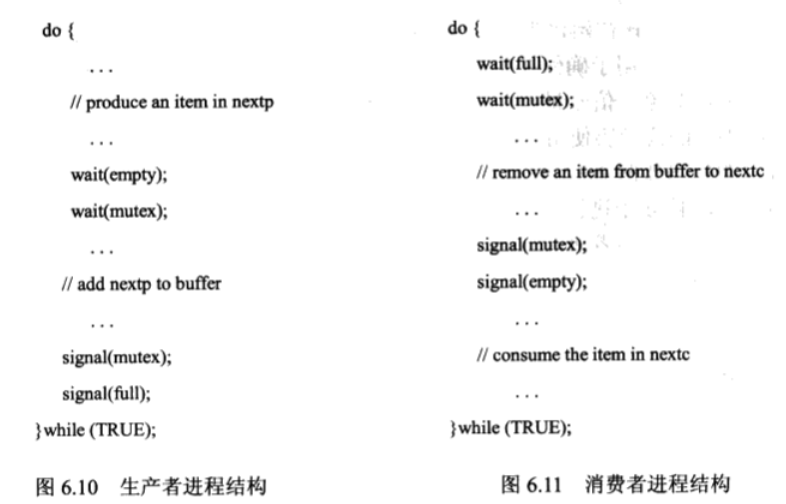


图1.1-1 生产者-消费者问题伪代码结构

1. **实验步骤：**

**2.1 表示缓冲区：**

问题中的缓冲区使用循环数组来组织，为了遵循书中实验问题中的设计，此处将循环数组的大小设置为5，循环数组项的类型设为，并使用头尾指针指向循环数组的首和尾。

只是使用这些变量来表示一个循环数组的话，由于数组为空时有，而数组为满时也有，为区分这两种情况，必须牺牲一个数组项空间来使得两种情况的指针指向情况不同。为此，我们引入一个变量来特定表示数组为满的情况，以节省下数组项空间。

**2.2 生产者与消费者函数：**

在这部分内容之后的内容中，由于Linux开发环境和Windows开发环境有所不同，但是基本思路都类似，因而代码差距并不大。在我们的设计思路中遵循教材实验的设计思路，将“进入/退出以及临界区代码”的设计抛送给插入/删除数据项的设计中，而在生产者和消费者的主体代码中实现剩余区的设计，在生产者的设计中，我们产生一个随机数作为生产出的项目，加入到缓冲区中，而消费者则负责向缓冲区中获取这个随机数并打印。在插入/删除数组项后我们都会额外地打印出缓冲区内容。

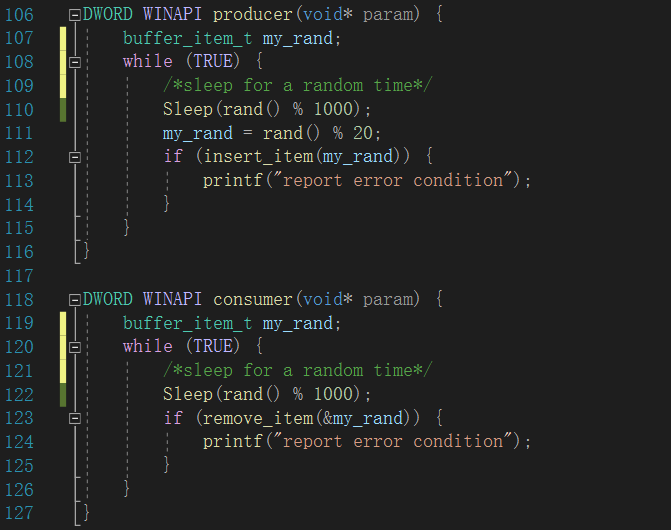


图2.2-1 生产者，消费者的实现

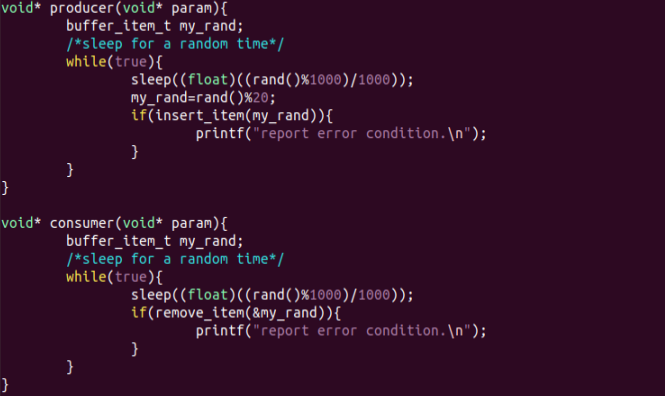


图2.2-2 生产者、消费者的实现

**2.3 临界区及其进入/退出部分的设计：**

依照相关伪代码结构的设计，可以获得相应对操作的代码，值得一提的是Windows和Linux对于线程和信号量的接口及其使用。

* + 1. **Windows的线程及信号量API：**

Windows的线程互斥锁使用的相关接口使用示例如下：

其中，是Windows中的句柄，实现为与对象一一对应的32无符号整型，Windows通过此来表示一个互斥锁或信号量。互斥锁初始化的原型为：

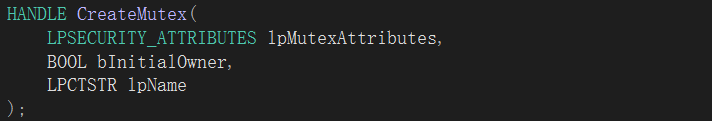


图2.3.1-1 原型

其中表示指向安全属性的指针，可设为；设置调用互斥对象的线程是否获得互斥对象的所有权；则是互斥锁的名称。而和则可以近似看为实现和操作的函数。

而Windows环境的信号量使用的相关示例如下：

其中信号量初始化原型为：

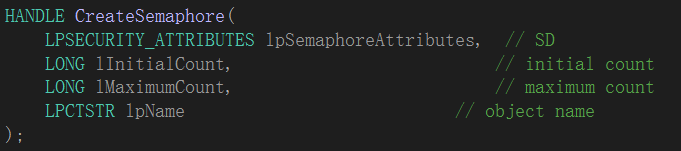


图2.3.1-2 原型

第一个参数和最后一个参数表示安全属性和信号量的名称，而第二、三个参数则表示信号的初值与最大值。同样的，和则可以近似看为实现和操作的函数。

* + 1. **Linux的线程及信号量API：**

在Linux中我们使用来实现相关的互斥和信号量操作，的相关操作使用示例如下：

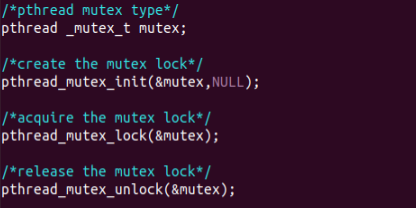


图2.3.2-1的使用示例

其中，在创建互斥锁时将第二个参数设置为可将互斥锁设置为默认值。当调用时，如果互斥锁不可用，那么调用线程会阻塞直到互斥锁占有为止。

而Linux的信号量也与互斥锁的操作基本相同，示例如下：

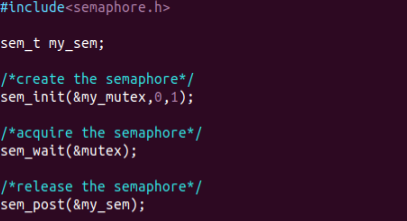


图2.3.2-2 示例

* + 1. **插入/删除数组项设计：**

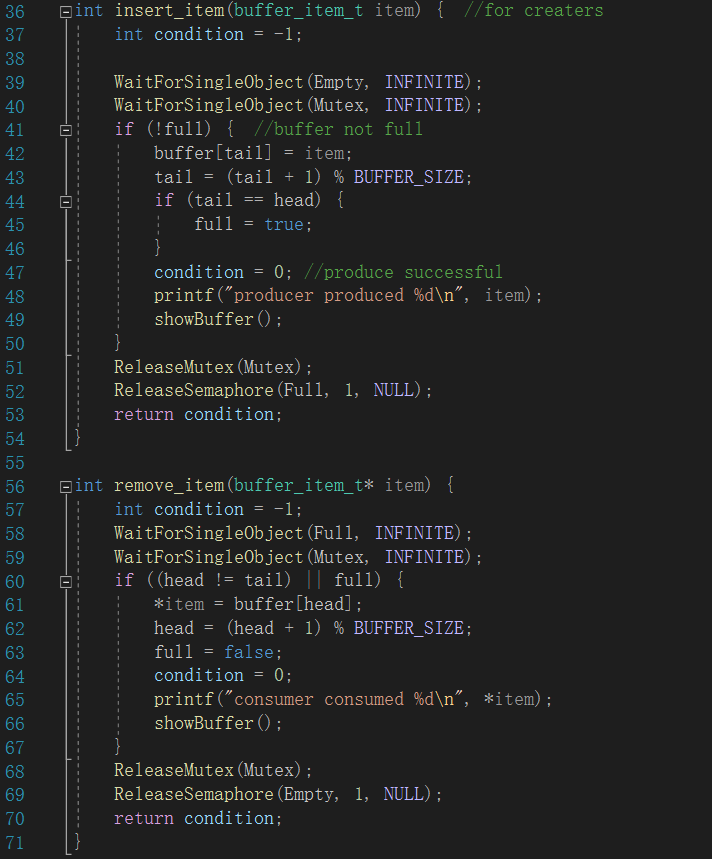


图2.3.3-1 Windows下设计

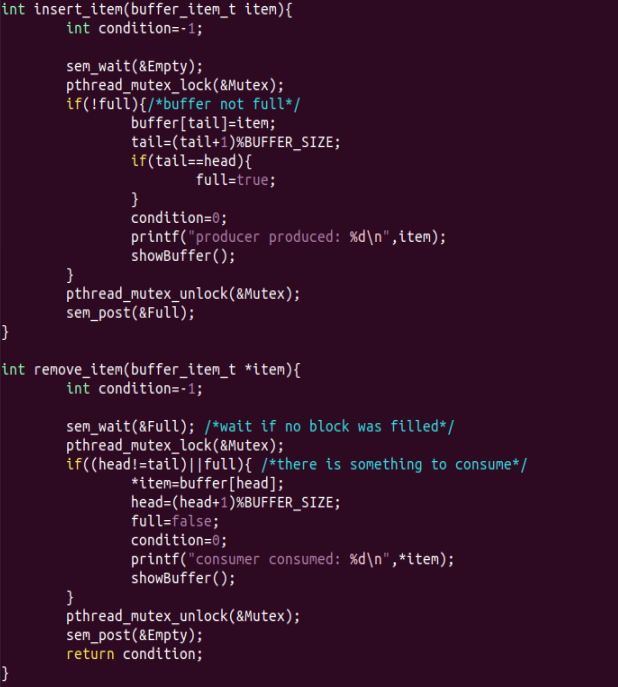
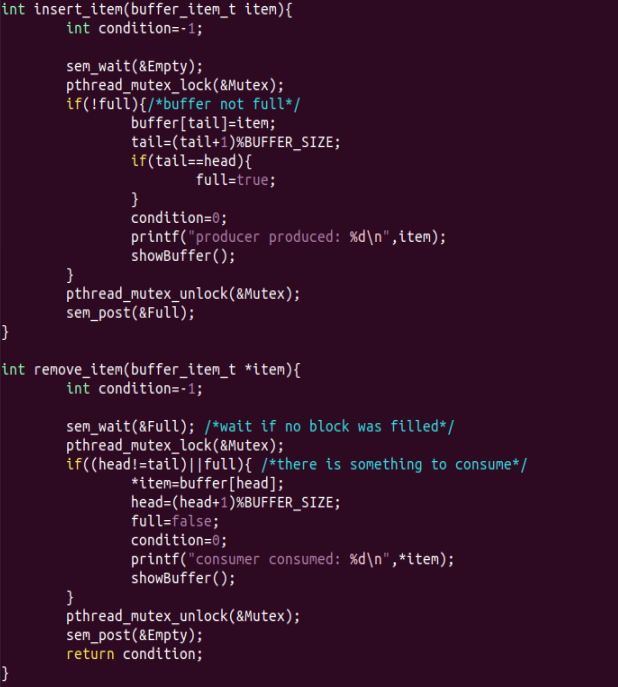
 

图2.3.3-2 linux下设计

**2.4 main函数设计：**

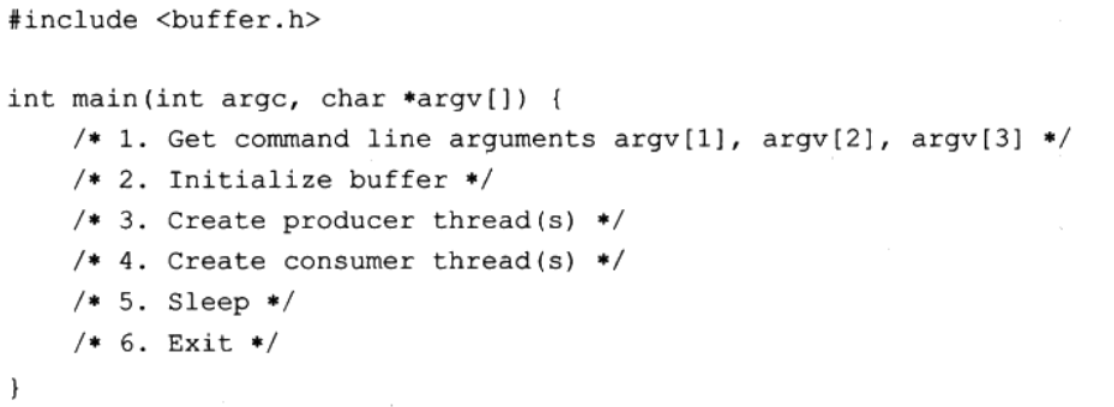
函数的整体结构将依照教材中对于函数的描述，即：

图2.4-1 函数结构

具体的实现如图所示：

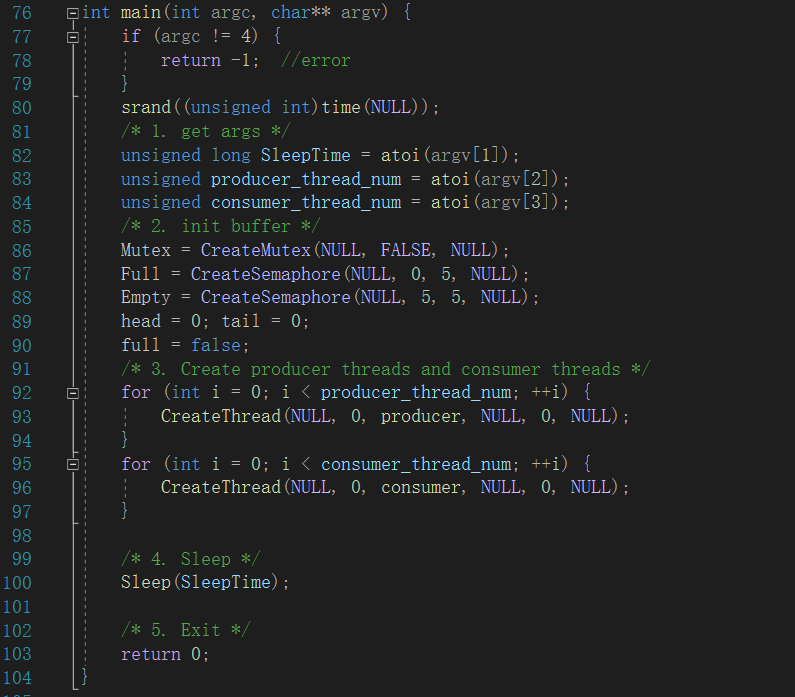
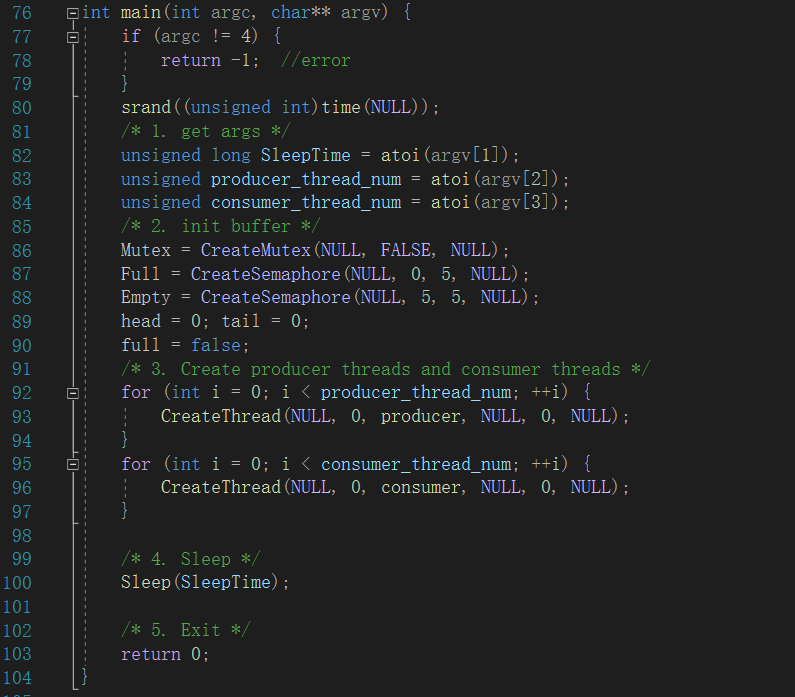
 

图2.4-2 Windows下函数设计



图2.4-2 Linux下函数设计

1. **运行结果测试**

**3.1 Windows环境下运行结果：**

Windows文件运行结果如下：

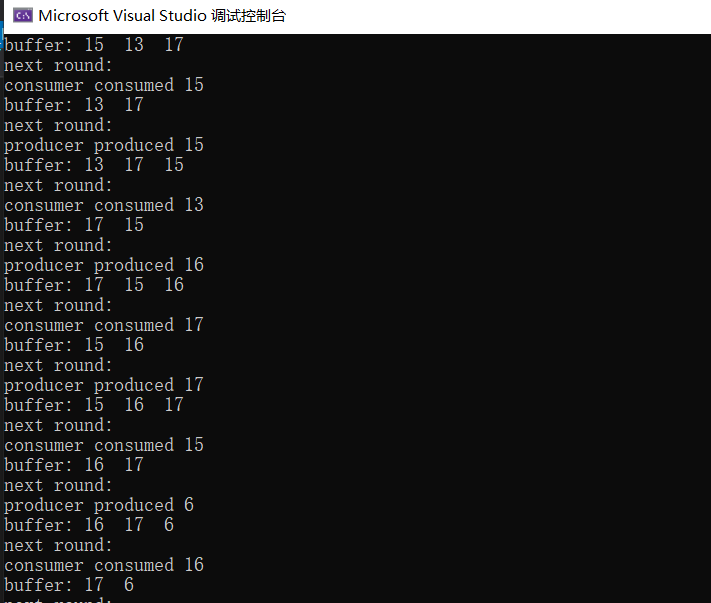


图1-1 Windows环境下部分运行结果

**3.2 Linux环境下运行结果：**

在Linux文件下使用gcc软件编译文件并执行的指令以及结果如下：



图2-1 Linux编译以及运行命令

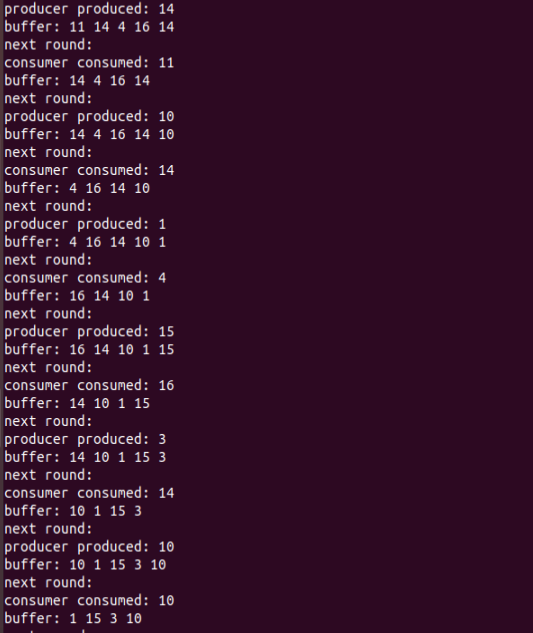
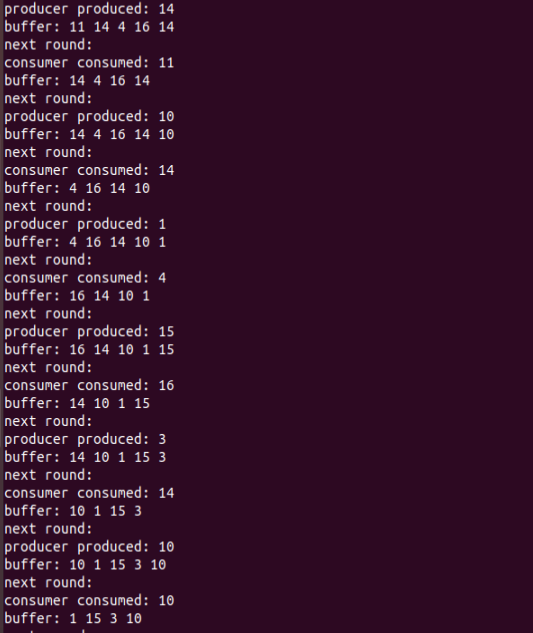
 

图2-2 Linux环境下部分运行结果

1. **实验小结**

**附：**

1. **Windows环境下相关源代码：**
2. #include<windows.h>
4. #define BUFFER\_SIZE 5
5. /\*buffer item type\*/
6. **typedef** **int** buffer\_item\_t;
7. /\*singnals\*/
8. /\*
9. \* Empty: empty "blocks" in buffer
10. \* Full: filled "blocks" in buffer
11. \*/
12. **HANDLE** Empty, Full, Mutex;
13. **size\_t** head, tail;
14. **bool** full;
16. /\*build buffer, which is acircular array\*/
17. buffer\_item\_t buffer[BUFFER\_SIZE];  /\*the buffer\*/
19. **void** showBuffer() {
20. printf("buffer: ");
21. **if** ((head != tail) || full) {
22. **int** cnt = head;
23. **do**{
24. printf("%d  ", buffer[cnt]);
25. cnt = (cnt + 1) % BUFFER\_SIZE;
26. } **while** (cnt != tail);
27. }
28. **else** {
29. printf("empty!");
30. }
31. printf("\nnext round:\n");
32. }
34. **int** insert\_item(buffer\_item\_t item) {  //for creaters
35. **int** condition = -1;
37. WaitForSingleObject(Empty, INFINITE);
38. WaitForSingleObject(Mutex, INFINITE);
39. **if** (!full) {  //buffer not full
40. buffer[tail] = item;
41. tail = (tail + 1) % BUFFER\_SIZE;
42. **if** (tail == head) {
43. full = **true**;
44. }
45. condition = 0; //produce successful
46. printf("producer produced %d\n", item);
47. showBuffer();
48. }
49. ReleaseMutex(Mutex);
50. ReleaseSemaphore(Full, 1, NULL);
51. **return** condition;
52. }
54. **int** remove\_item(buffer\_item\_t\* item) {
55. **int** condition = -1;
56. WaitForSingleObject(Full, INFINITE);
57. WaitForSingleObject(Mutex, INFINITE);
58. **if** ((head != tail) || full) {
59. \*item = buffer[head];
60. head = (head + 1) % BUFFER\_SIZE;
61. full = **false**;
62. condition = 0;
63. printf("consumer consumed %d\n", \*item);
64. showBuffer();
65. }
66. ReleaseMutex(Mutex);
67. ReleaseSemaphore(Empty, 1, NULL);
68. **return** condition;
69. }
71. **DWORD** WINAPI producer(**void**\* param);
72. **DWORD** WINAPI consumer(**void**\* param);
74. **int** main(**int** argc, **char**\*\* argv) {
75. **if** (argc != 4) {
76. **return** -1;  //error
77. }
78. srand((unsigned **int**)time(NULL));
79. /\* 1. get args \*/
80. unsigned **long** SleepTime = atoi(argv[1]);
81. unsigned producer\_thread\_num = atoi(argv[2]);
82. unsigned consumer\_thread\_num = atoi(argv[3]);
83. /\* 2. init buffer \*/
84. Mutex = CreateMutex(NULL, FALSE, NULL);
85. Full = CreateSemaphore(NULL, 0, 5, NULL);
86. Empty = CreateSemaphore(NULL, 5, 5, NULL);
87. head = 0; tail = 0;
88. full = **false**;
89. /\* 3. Create producer threads and consumer threads \*/
90. **for** (**int** i = 0; i < producer\_thread\_num; ++i) {
91. CreateThread(NULL, 0, producer, NULL, 0, NULL);
92. }
93. **for** (**int** i = 0; i < consumer\_thread\_num; ++i) {
94. CreateThread(NULL, 0, consumer, NULL, 0, NULL);
95. }
97. /\* 4. Sleep \*/
98. Sleep(SleepTime);
100. /\* 5. Exit \*/
101. **return** 0;
102. }
104. **DWORD** WINAPI producer(**void**\* param) {
105. buffer\_item\_t my\_rand;
106. **while** (TRUE) {
107. /\*sleep for a random time\*/
108. Sleep(rand() % 1000);
109. my\_rand = rand() % 20;
110. **if** (insert\_item(my\_rand)) {
111. printf("report error condition");
112. }
113. }
114. }
116. **DWORD** WINAPI consumer(**void**\* param) {
117. buffer\_item\_t my\_rand;
118. **while** (TRUE) {
119. /\*sleep for a random time\*/
120. Sleep(rand() % 1000);
121. **if** (remove\_item(&my\_rand)) {
122. printf("report error condition");
123. }
124. }
125. }

**2. Linux环境下相关源代码：**

1. #include<semaphore.h>
2. #include<pthread.h>
3. #include<unistd.h>
4. #include<time.h>
5. #include<stdlib.h>
6. #include<stdio.h>
8. /\*fixed buffer size\*/
9. #define BUFFER\_SIZE 5
10. /\*buffer item type\*/
11. **typedef** **int** buffer\_item\_t;
13. /\*init semaphore\*/
14. pthread\_mutex\_t Mutex;
16. sem\_t Full,Empty;
17. **size\_t** head,tail;
18. /\*bool to indicate whether buffer is full\*/
19. **bool** full;
21. /\*build buffer, thich is a circular array\*/
22. buffer\_item\_t buffer[BUFFER\_SIZE];
24. **void** showBuffer(){
25. printf("buffer: ");
26. **if**((head!=tail)||full){
27. **int** cnt=head;
28. **do**{
29. printf("%d ",buffer[cnt]);
30. cnt=(cnt+1)%BUFFER\_SIZE;
31. }**while**(cnt!=tail);
32. }**else**{
33. printf("empty");
34. }
35. printf("\nnext round: \n");
36. }
38. **int** insert\_item(buffer\_item\_t item){
39. **int** condition=-1;
41. sem\_wait(&Empty);
42. pthread\_mutex\_lock(&Mutex);
43. **if**(!full){/\*buffer not full\*/
44. buffer[tail]=item;
45. tail=(tail+1)%BUFFER\_SIZE;
46. **if**(tail==head){
47. full=**true**;
48. }
49. condition=0;
50. printf("producer produced: %d\n",item);
51. showBuffer();
52. }
53. pthread\_mutex\_unlock(&Mutex);
54. sem\_post(&Full);
55. }
57. **int** remove\_item(buffer\_item\_t \*item){
58. **int** condition=-1;
60. sem\_wait(&Full); /\*wait if no block was filled\*/
61. pthread\_mutex\_lock(&Mutex);
62. **if**((head!=tail)||full){ /\*there is something to consume\*/
63. \*item=buffer[head];
64. head=(head+1)%BUFFER\_SIZE;
65. full=**false**;
66. condition=0;
67. printf("consumer consumed: %d\n",\*item);
68. showBuffer();
69. }
70. pthread\_mutex\_unlock(&Mutex);
71. sem\_post(&Empty);
72. **return** condition;
73. }
75. **void**\* producer(**void**\* param);
76. **void**\* consumer(**void**\* param);
78. **int** main(**int** argc, **char**\*\*argv){
79. **if**(argc!=4){
80. **return** -1;
81. }
82. srand((unsigned **int**)time(NULL));
83. /\* 1. get args \*/
84. unsigned **long** SleepTime=atol(argv[1]);
85. unsigned producer\_thread\_num=atoi(argv[2]);
86. unsigned consumer\_thread\_num=atoi(argv[3]);
87. printf("%ld, %d, %d\n",SleepTime,producer\_thread\_num,consumer\_thread\_num);
88. /\* 2. init buffer \*/
89. head=0;tail=0;
90. full=**false**;
91. pthread\_mutex\_init(&Mutex,NULL);
92. sem\_init(&Full,0,0);
93. sem\_init(&Empty,0,BUFFER\_SIZE);
94. /\* 3. Create producer threads and consumer threads \*/
95. pthread\_attr\_t my\_attr;
96. pthread\_attr\_init(&my\_attr);
97. **for**(unsigned i=0;i<producer\_thread\_num;++i){
98. pthread\_t producer\_thread;
99. pthread\_create(&producer\_thread,&my\_attr,producer,NULL);
100. }
101. **for**(unsigned i=0;i<consumer\_thread\_num;++i){
102. pthread\_t consumer\_thread;
103. pthread\_create(&consumer\_thread,&my\_attr,consumer,NULL);
104. }
106. /\* 4. Sleep \*/
107. sleep((**float**)(SleepTime/1000));
109. /\* Exit \*/
110. **return** 0;
111. }
113. **void**\* producer(**void**\* param){
114. buffer\_item\_t my\_rand;
115. /\*sleep for a random time\*/
116. **while**(**true**){
117. sleep((**float**)((rand()%1000)/1000));
118. my\_rand=rand()%20;
119. **if**(insert\_item(my\_rand)){
120. printf("report error condition.\n");
121. }
122. }
123. }
125. **void**\* consumer(**void**\* param){
126. buffer\_item\_t my\_rand;
127. /\*sleep for a random time\*/
128. **while**(**true**){
129. sleep((**float**)((rand()%1000)/1000));
130. **if**(remove\_item(&my\_rand)){
131. printf("report error condition.\n");
132. }
133. }
134. }