Producer-Consumer problem实验报告

57119101 王晨阳

2021年6月1日

实验目的

通过实验，理解Win32 API、Pthreads中mutex locks、semaphores等使用方法，并掌握如何利用它们实现进程（线程）间的同步和互斥。

实验步骤

Win32 API

程序

```
/*********************************************************************************
  *Copyright(C),Chenyang
  *FileName: ProducerConsumerWin32.c
  *Author: 王晨阳
  *Date: 2021-06-01
  *Description:
        Using Win32 api to accomplish the task of PRODUCER_CONSUMER_PROBLEM.
        Input should be like [number of producer]
                             [number of consumer]
                             [number of items every producer produced]
                             [number of items every consumer consumed].
*********************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

```

```
typedef int buffer_item;
#define BUFFER_SIZE 5                    //buffer can contain 5 items
#define MAX_SLEEP_TIME 500               //sleep for 500 ms

HANDLE full, empty, mutex;               //semaphore & mutex

buffer_item buffer[BUFFER_SIZE];         //a circular queue
int top, rear;                           //head and tail of the circular queue

int ProducerCount,     ConsumerCount;    //number of producers/consumers
int ProducerItemCount, ConsumerItemCount; //number of items every producer produced/consumer consumed

int insert_item(buffer_item item)
{
    WaitForSingleObject(empty, INFINITE);
    WaitForSingleObject(mutex, INFINITE);

    //critical section
    buffer[rear] = item;
    rear = (rear == BUFFER_SIZE - 1) ? 0 : rear + 1;

    ReleaseMutex(mutex);
    ReleaseSemaphore(full,  1, NULL);

    return 0;
}

int remove_item(buffer_item *item)
{
    WaitForSingleObject(full,  INFINITE);
    WaitForSingleObject(mutex, INFINITE);

    //critical section
    *item = buffer[top];
    top = (top == BUFFER_SIZE - 1) ? 0 : top + 1;

    ReleaseMutex(mutex);
    ReleaseSemaphore(empty, 1, NULL);

    return 0;
}

DWORD WINAPI producer(LPVOID Param)
{
    DWORD User = *(DWORD *)Param;
    srand(time(NULL) + User); //ensure randomness
    for (DWORD i = 1; i <= ProducerItemCount; i++)
    {
        Sleep(rand() % MAX_SLEEP_TIME + 1);
        buffer_item item = rand();
        if (insert_item(item))
            fprintf(stderr, "report error condition\n");
        else
            printf("producer %d produced %d\n", User, item);
    }
}

```

```
76  DWORD WINAPI consumer(LPVOID Param)
77  {
78      DWORD User = *(DWORD *)Param;
79      srand(time(NULL) + User + ProducerCount); //ensure randomness
80      for (DWORD i = 1; i <= ConsumerItemCount; i++)
81      {
82          Sleep(rand() % MAX_SLEEP_TIME + 1);
83          buffer_item item;
84          if (remove_item(&item))
85              fprintf(stderr, "report error condition\n");
86          else
87              printf("consumer %d consumed %d\n", User, item);
88      }
89  }
90
91  int main(int argc, char *argv[])
92  {
93      //check for parameter
94      if (argc != 5)
95      {
96          fprintf(stderr, "Parameter is required\n");
97          return -1;
98      }
99
100     ProducerCount     = atoi(argv[1]);
101     ConsumerCount     = atoi(argv[2]);
102     ProducerItemCount = atoi(argv[3]);
103     ConsumerItemCount = atoi(argv[4]);
104
105     if (ProducerCount * ProducerItemCount < ConsumerCount * ConsumerItemCount)
106     {
107         fprintf(stderr, "There should be more items produced than consumed\n");
108         return -1;
109     }
110
111     //initialize
112     memset(buffer, 0, sizeof(buffer));
113     top = rear = 0;
114
115     //create semaphore
116     full  = CreateSemaphore(NULL, 0, BUFFER_SIZE, NULL);
117     empty = CreateSemaphore(NULL, BUFFER_SIZE, BUFFER_SIZE, NULL);
118     mutex = CreateMutex(NULL, FALSE, NULL);
119
120     //create thread
121     DWORD  ProducerId [ProducerCount];
122     DWORD  ConsumerId [ConsumerCount];
123     HANDLE PRODUCER   [ProducerCount];
124     HANDLE CONSUMER   [ConsumerCount];
125     DWORD  CurProducer[ProducerCount]; //record the producer/consumer the thread
    belongs to
126     DWORD  CurConsumer[ConsumerCount];
127
128     for (DWORD i = 0; i < ProducerCount; i++)
129     {
130         CurProducer[i] = i;
131         PRODUCER[i] = CreateThread(NULL, 0, producer, &CurProducer[i], 0,
    &ProducerId[i]);
```

```
132         }
133         for (DWORD i = 0; i < ConsumerCount; i++)
134         {
135             CurConsumer[i] = i;
136             CONSUMER[i] = CreateThread(NULL, 0, consumer, &CurConsumer[i], 0,
    &ConsumerId[i]);
137         }
138 
139         //terminate thread
140         for (DWORD i = 0; i < ProducerCount; i++)
141             if (PRODUCER[i] != NULL)
142             {
143                 WaitForSingleObject(PRODUCER[i], INFINITE);
144                 CloseHandle(PRODUCER[i]);
145             }
146         for (DWORD i = 0; i < ConsumerCount; i++)
147             if (CONSUMER[i] != NULL)
148             {
149                 WaitForSingleObject(CONSUMER, INFINITE);
150                 CloseHandle(CONSUMER);
151             }
152 
153         return 0;
154     }
```

运行结果

> SYSTEM: Windows10 21H2

```
1   PS> gcc ProducerConsumerWin32.c -o ProducerConsumerWin32
2   PS> ./ProducerConsumerWin32 5 5 4 4
```

```
1   producer 2 produced 11459
2   consumer 4 consumed 11459
3   producer 1 produced 711
4   producer 0 produced 22730
5   producer 3 produced 22208
6   consumer 3 consumed 711
7   consumer 1 consumed 22730
8   producer 4 produced 188
9   consumer 0 consumed 22208
10  consumer 2 consumed 188
11  producer 2 produced 3150
12  consumer 4 consumed 3150
13  producer 0 produced 20559
14  consumer 1 consumed 20559
15  producer 0 produced 22627
16  producer 3 produced 27213
17  producer 1 produced 11854
18  producer 2 produced 30481
19  consumer 3 consumed 22627
20  consumer 2 consumed 27213
21  producer 4 produced 18508
22  consumer 0 consumed 11854
23  consumer 4 consumed 30481
24  producer 2 produced 9851
```

```
producer 0 produced 15706
consumer 1 consumed 18508
producer 1 produced 26554
consumer 2 consumed 9851
consumer 4 consumed 15706
consumer 0 consumed 26554
producer 3 produced 1639
consumer 3 consumed 1639
producer 4 produced 5566
consumer 1 consumed 5566
producer 3 produced 6924
consumer 2 consumed 6924
producer 1 produced 12778
consumer 0 consumed 12778
producer 4 produced 3996
consumer 3 consumed 3996
```

Pthreads

程序

```c
/*********************************************************************************
  *Copyright(C),Chenyang
  *FileName: ProducerConsumerPthread.c
  *Author: 王晨阳
  *Date: 2021-06-01
  *Description:
        Using Pthread to accomplish the task of PRODUCER_CONSUMER_PROBLEM.
        Input should be like [number of producer]
                             [number of consumer]
                             [number of items every producer produced]
                             [number of items every consumer consumed].
*********************************************************************************/

#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

typedef int buffer_item;
#define BUFFER_SIZE 5                   //buffer can contain 5 items
#define MAX_SLEEP_TIME 500              //sleep for 500 ms

sem_t full, empty;                      //semaphore
pthread_mutex_t mutex;                  //mutex

buffer_item buffer[BUFFER_SIZE];        //a circular queue
int top, rear;                          //head and tail of the circular queue

int ProducerCount, ConsumerCount;       //number of producer/consumer
```

```c
int ProducerItemCount, ConsumerItemCount; //number of items every producer produced/consumer consumed

int insert_item(buffer_item item)
{
    sem_wait(&empty);
    pthread_mutex_lock(&mutex);

    //critical section
    buffer[rear] = item;
    rear = (rear == BUFFER_SIZE - 1) ? 0 : rear + 1;

    pthread_mutex_unlock(&mutex);
    sem_post(&full);

    return 0;
}

int remove_item(buffer_item *item)
{
    sem_wait(&full);
    pthread_mutex_lock(&mutex);

    //critical section
    *item = buffer[top];
    top = (top == BUFFER_SIZE - 1) ? 0 : top + 1;

    pthread_mutex_unlock(&mutex);
    sem_post(&empty);

    return 0;
}

void *producer(void *Param)
{
    int User = *(int *)Param;
    srand(time(NULL) + User); //ensure randomness
    for (int i = 1; i <= ProducerItemCount; i++)
    {
        sleep((double)(rand() % MAX_SLEEP_TIME + 1) / 1000.000);
        buffer_item item = rand();
        if (insert_item(item))
            fprintf(stderr, "report error condition\n");
        else
            printf("producer %d produced %d\n", User, item);
    }
}

void *consumer(void *Param)
{
    int User = *(int *)Param;
    srand(time(NULL) + User + ProducerCount); //ensure randomness
    for (int i = 1; i <= ConsumerItemCount; i++)
    {
        sleep((double)(rand() % MAX_SLEEP_TIME + 1) / 1000.000);
        buffer_item item;
        if (remove_item(&item))
            fprintf(stderr, "report error condition\n");
```

```
90          else
91              printf("consumer %d consumed %d\n", User, item);
92      }
93  }
94
95  int main(int argc, char *argv[])
96  {
97      //check for parameter
98      if (argc != 5)
99      {
100         fprintf(stderr, "Parameter is required\n");
101         return -1;
102     }
103     ProducerCount     = atoi(argv[1]);
104     ConsumerCount     = atoi(argv[2]);
105     ProducerItemCount = atoi(argv[3]);
106     ConsumerItemCount = atoi(argv[4]);
107     if (ProducerCount * ProducerItemCount < ConsumerCount * ConsumerItemCount)
108     {
109         fprintf(stderr, "There should be more items produced than consumed\n");
110         return -1;
111     }
112
113     //initialize
114     memset(buffer, 0, sizeof(buffer));
115     top = rear = 0;
116
117     //create semaphore
118     sem_init(&full, 0, 0);
119     sem_init(&empty, 0, BUFFER_SIZE);
120     pthread_mutex_init(&mutex, NULL);
121
122     //create thread
123     pthread_t PRODUCER[ProducerCount];
124     pthread_t CONSUMER[ConsumerCount];
125     int CurProducer[ProducerCount]; //record the producer/consumer the thread
    belongs to
126     int CurConsumer[ConsumerCount];
127
128     for (int i = 0; i < ProducerCount; i++)
129     {
130         CurProducer[i] = i;
131         pthread_create(&PRODUCER[i], NULL, producer, &CurProducer[i]);
132     }
133     for (int i = 0; i < ConsumerCount; i++)
134     {
135         CurConsumer[i] = i;
136         pthread_create(&CONSUMER[i], NULL, consumer, &CurConsumer[i]);
137     }
138
139     //terminate thread
140     for (int i = 0; i < ProducerCount; i++)
141         pthread_join(PRODUCER[i], NULL);
142     for (int i = 0; i < ConsumerCount; i++)
143         pthread_join(CONSUMER[i], NULL);
144
145     return 0;
146 }
```

运行结果

> SYSTEM: Ubuntu20.04 (WSL)

```
$ gcc ProducerConsumerPthread.c -o ProducerConsumerPthread -lpthread
$ ./ProducerConsumerPthread 5 5 4 4
```

```
producer 1 produced 1823051570
producer 0 produced 1641975433
producer 3 produced 34636998
consumer 0 consumed 1823051570
consumer 1 consumed 34636998
producer 3 produced 1964985380
consumer 2 consumed 1641975433
producer 2 produced 433018536
consumer 4 consumed 433018536
consumer 1 consumed 1592485399
consumer 2 consumed 145032469
producer 2 produced 1608767659
producer 0 produced 145032469
consumer 1 consumed 1362920205
consumer 2 consumed 1608767659
consumer 0 consumed 413485799
consumer 2 consumed 1275366272
producer 2 produced 1275366272
producer 4 produced 1592485399
consumer 3 consumed 1964985380
consumer 4 consumed 867469038
producer 3 produced 1362920205
producer 4 produced 223031996
producer 0 produced 170437731
producer 4 produced 1589508306
consumer 1 consumed 170437731
producer 2 produced 867469038
consumer 0 consumed 223031996
producer 1 produced 413485799
producer 0 produced 933701033
producer 4 produced 1127811455
consumer 0 consumed 1581603209
consumer 4 consumed 933701033
producer 3 produced 1581603209
producer 1 produced 896235234
consumer 3 consumed 1589508306
consumer 4 consumed 1127811455
producer 1 produced 1626311734
consumer 3 consumed 896235234
consumer 3 consumed 1626311734
```

实验体会

通过本次实验，掌握了 Win32 API 和 Pthreads 的基本使用，实践了 Producer-Consumer problem 的解决方法，加深了对操作系统概念的理解。提高了动手能力，解决问题的能力得到强化。