Project 5-2: The Producer-Consumer Problem

白骐硕 518030910102

任务概述

In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in Figures 5.9 and 5.10. The solution presented in Section 7.1.1 uses three semaphores: empty and full , which count the number of empty and full slots in the buffer, and mutex , which is a binary (or mutual exclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, you will use standard counting semaphores for empty and full and a mutex lock, rather than a binary semaphore, to represent mutex . The producer and consumer—running as separate threads—will move items to and from a buffer that is synchronized with the empty , full , and mutex structures. You can solve this problem using either Pthreads or the Windows API .

具体实现

使用Pthreads完成本实验。本实验中需要编写的文件共有3个：

- buffer.h
- buffer.c
- main.c

下面，依次对其进行介绍。

buffer.h

首先在buffer.h中定义需要用到的常量：

- BUFFER_SIZE
- MAX_SLEEP_TIME，限定了producer和consumer随机sleep 的时间长度
- MAX_ITEM，限定了 item的最大值

其次，定义类型 buffer_iem

```
typedef int buffer_item;
```

最后，声明在buffer.c中需要实现的函数

```
int insert_item(buffer_item item);
int remove_item(buffer_item *item);
void *consumer(void *param);
void *producer(void *param);
void buffer_init();
void buffer_shutdown();
```

buffer.c

在该部分，我将详细介绍上述在buffer.h中声明的若干函数，在这之前，先介绍在buffer.c中定义的若干全局变量，分为三个部分：

- buffer
- semaphore
- mutex

```
//buffer
buffer_item buffer[BUFFER_SIZE];
int bufferHead,bufferTail;
//semaphore
sem_t full,empty;
//mutex
pthread_mutex_t lock;
```

下面详细介绍需要实现的函数：

insert_item() remove_item()

执行逻辑为：

- 首先 sem_wait(&empty)
- 之后将mutex加锁，以便进行后续对buffer的操作
- 在buffer的尾部添加item
- 释放mutex
- sem_post(&full)

```
int insert_item(buffer_item item) {
    sem_wait(&empty);
    pthread_mutex_lock(&lock);
    buffer[bufferTail] = item;
    bufferTail = (bufferTail + 1) %(BUFFER_SIZE);
    pthread_mutex_unlock(&lock);
    sem_post(&full);
    return 0;
}
```

remove的执行逻辑与insert相同，只是调换empty 和full的位置，且将插入改为删除即可。

```
int remove_item(buffer_item *item) {
    sem_wait(&full);
    pthread_mutex_lock(&lock);
    *item = buffer[bufferHead];
    bufferHead = (bufferHead+1) % BUFFER_SIZE;
    pthread_mutex_unlock(&lock);
    sem_post(&empty);
    return 0;
}
```

producer() consumer()

producer() 为生产者线程的工作函数：

- 随机sleep一段时间（受限于所定义的常量 MAX_SLEEP_TIME）
- 随机产生一个item（受限于所定义的常量 MAX_ITEM）
- 将该item插入的buffer中
- 打印执行过程信息

consumer() 的内部逻辑与producer()互相对应。

```
void *producer(void *param) {
    buffer_item item;
    while (1) {
        sleep(rand()% MAX_SLEEP_TIME+1);
        item = rand() % MAX_ITEM;
        if(insert_item(item)) {
            fprintf(stderr,"Error: produce failed!");
        } else {
            printf("The Producer %d produces the value %d. \n" , *(int*) param,
item);
        }
    }
}

void *consumer(void *param) {
    buffer_item item;
    while(1) {
        sleep(rand()% MAX_SLEEP_TIME+1);
        if(remove_item(&item)) {
            fprintf(stderr,"Error: consume failed!");
        } else {
            printf("The Consumer %d consumes the value %d. --Consumer\n" , *
(int*) param, item);
        }
    }
}
```

buffer_init() buffer_shutdown()

需要初始化的内容为：

- mutex
- 两个semaphore
- bufferHead，bufferTail两个用于标记buffer中item首位的int

```
//init
void buffer_init() {
    pthread_mutex_init(&lock,NULL);
    sem_init(&full,0,0);
    sem_init(&empty,0,BUFFER_SIZE);
    bufferHead = 0;
    bufferTail = 0;
}
```

需要删除（destroy）的内容为：

- mutex
- 两个semaphore

```
//destroy
void buffer_shutdown() {
    sem_destroy(&full);
    sem_destroy(&empty);
    pthread_mutex_destroy(&lock);
}
```

main.c

下面介绍main函数的编写，依照课本中的提示内容进行编写：

```
#include "buffer.h"

int main(int argc, char *argv[]) {
  /* 1. Get command line arguments argv[1],argv[2],argv[3] */
  /* 2. Initialize buffer */
  /* 3. Create producer thread(s) */
  /* 4. Create consumer thread(s) */
  /* 5. Sleep */
  /* 6. Exit */
}
```

```
int main(int argc, char *argv[]) {
    // Get the command line arguments
    if(argc != 4) {
        fprintf(stderr,"Usage: <executable> sleepSecond numProducer numConsumer
\n");
    }
    int sleep_time = atoi(argv[1]);
    int num_producer = atoi(argv[2]), num_consumer = atoi(argv[3]);

    // Initialize buffer
    buffer_init();

    // Create producer threads
    pthread_t *producers = malloc(num_producer * sizeof(pthread_t));
    int producer_id[num_producer];
    for (int i = 0; i < num_producer; ++i) {
        producer_id[i] = i+1;
        pthread_create(&producers[i],NULL,producer,&producer_id[i]);
    }

    // Create consumer threads
    pthread_t *consumers = malloc(num_consumer * sizeof(pthread_t));
    int consumer_id[num_consumer];
    for (int i = 0; i < num_producer; ++i) {
        consumer_id[i] = i+1;
        pthread_create(&consumers[i],NULL,consumer,&consumer_id[i]);
    }
```

```
    // Sleep
    printf("Sleep for %d second(s) before exit.\n", sleep_time);
    sleep(sleep_time);

    // Exit
    // cancel producer
    for(int i = 0; i != num_producer; ++i) {
        pthread_cancel(producers[i]);
        pthread_join(producers[i], NULL);
    }
    //cancel consumer
    for(int i = 0; i != num_consumer; ++i) {
        pthread_cancel(consumers[i]);
        pthread_join(consumers[i], NULL);
    }
    free(producers);
    free(consumers);
    buffer_shutdown();

    return 0;
}
```

实验结果

sleepSecond = 5, number of producer = 5, number of consumer = 5 的执行结果如下：

```
qsbai@qsbai-virtual-machine:~/os-prj/5/5-2$ ./main 5 5 5
Sleep for 5 second(s) before exit.
The Producer 3 produces the value 62.
The Consumer 4 consumes the value 62. --Consumer
The Producer 2 produces the value 59.
The Consumer 1 consumes the value 59. --Consumer
The Consumer 3 consumes the value 40. --Consumer
The Producer 1 produces the value 40.
The Producer 4 produces the value 36.
The Consumer 2 consumes the value 36. --Consumer
The Producer 5 produces the value 67.
The Consumer 5 consumes the value 67. --Consumer
The Producer 3 produces the value 30.
The Consumer 3 consumes the value 30. --Consumer
The Producer 5 produces the value 67.
The Consumer 4 consumes the value 67. --Consumer
The Producer 1 produces the value 2.
The Consumer 1 consumes the value 2. --Consumer
The Producer 2 produces the value 69.
The Consumer 2 consumes the value 69. --Consumer
The Producer 4 produces the value 56.
The Consumer 3 consumes the value 56. --Consumer
```

sleepSecond = 5, number of producer = 8, number of consumer = 8 的执行结果如下：

```
qsbai@qsbai-virtual-machine:~/os-prj/5/5-2$ ./main 5 8 8
Sleep for 5 second(s) before exit.
The Producer 3 produces the value 40.
The Producer 8 produces the value 72.
The Consumer 1 consumes the value 40. --Consumer
The Producer 2 produces the value 67.
The Consumer 2 consumes the value 72. --Consumer
The Consumer 4 consumes the value 67. --Consumer
The Producer 4 produces the value 23.
The Consumer 6 consumes the value 29. --Consumer
The Producer 6 produces the value 82.
The Producer 7 produces the value 30.
The Producer 1 produces the value 29.
The Consumer 8 consumes the value 68. --Consumer
The Producer 3 produces the value 68.
The Producer 5 produces the value 56.
The Producer 8 produces the value 42.
The Consumer 7 consumes the value 82. --Consumer
The Consumer 8 consumes the value 42. --Consumer
The Consumer 3 consumes the value 30. --Consumer
The Consumer 5 consumes the value 23. --Consumer
The Producer 7 produces the value 29.
The Producer 6 produces the value 73.
The Consumer 6 consumes the value 56. --Consumer
The Producer 3 produces the value 13.
The Consumer 1 consumes the value 13. --Consumer
The Consumer 4 consumes the value 29. --Consumer
The Consumer 2 consumes the value 73. --Consumer
The Producer 8 produces the value 73.
The Consumer 6 consumes the value 73. --Consumer
The Producer 5 produces the value 96.
The Producer 4 produces the value 81.
The Producer 1 produces the value 5.
The Producer 2 produces the value 25.
The Consumer 3 consumes the value 96. --Consumer
```