操作系统课程设计报告

生产者-消费者问题

1 项目简介

1.1 项目目的

- 学习基于信号量的有限缓冲下的生产者消费者问题。

1.2 项目要求

使用三个信号量：empty（记录有多少空位），full（记录有多少满位），muxtex（二进制信号量或互斥信号量，以保护对缓冲插入与删除的操作）解决有限缓冲下的生产者消费者问题。

2 项目实现

2.1 系统版本

- Ubuntu：16.04
- Linux内核：4.15.13

2.2 实现过程

2.2.1 缓冲区

缓冲区的元数据类型为buffer_item的固定大小数组，在本实验中使用typedef将数据类型定义为int，并定义其大小为5.

```
#define BUFFER_SIZE 5

typedef int buffer_item;

buffer_item buffer[BUFFER_SIZE];
```

图 1: 缓冲区定义

并全局定义pthread_mutex_t类型的互斥锁mutex，sem_t类型的信号量full,empty，以及pthread_t类型的线程id在后续主函数初始化，生成线程以及线程函数中使用。

```
pthread_mutex_t mutex;
sem_t full, empty;
pthread_t tid;
```

图 2: 缓冲区定义

缓冲区操作需要通过两个函数实现：insert_item()和remove_item()。这两个函数分别被生产者、消费者线程在临界区内使用：

```
int insert_item(buffer_item item)
{
   /* insert item into buffer */
    if(p < BUFFER_SIZE - 1)
    {
        p++;
        buffer[p] = item;
        return 0;
    }
    else
        return -1;
}


int remove_item(buffer_item *item)
{
   /* insert item into buffer */
    if(p >= 0)
    {
        *item = buffer[p];
        p--;
        return 0;
    }
    else
        return -1;
}
```

图 3: 缓冲区使用函数

其中p作为指向当前最后一个数据的指针，使用if函数判断当前缓冲区是否为满（空），若生产者试图向满缓冲区插入数据（消费者试图向空缓冲区取出数据）则返回-1表示错误，否则执行插入（取出）并更新p值。

2.2.2 生产者消费者线程

生产者（消费者）线程不断交替执行两个阶段：睡眠随机时间，向缓冲区插入（取出）一个数据。

其中rand()用于产生一个[0,RAND_MAX]内的随机数，通过rand()%n产生$[0, n)$内的随机数作为睡眠时间和生成数。

对缓冲区插入/取出数据作为临界区，受到信号量full（消费者），empty（生产者）和互斥锁mutex的保护。对于信号量，Pthread中wait()和signal()分别对应sem_wait()和sem_post()。对于互斥锁,Pthread中获得锁的函数为pthread_mutex_lock()，释放锁的函数为pthread_mutex_unlock()。应当先获取信号量再获取互斥锁以进入临界区，出临界区后先释放互斥锁再释放信号量。

因此，临界区操作总过程为：

/*acquire the semaphore*/
sem_wait(&semaphore);
/*acquire the mutex lock*/
pthread_mutex_lock(&mutex);
/***critical section***/
/*release the mutex lock*/
pthread_mutex_unlock(&mutex);
/*release the semaphore*/
sem_post(&semaphore);

```c
void *producer(void *var)
{
    buffer_item item;

    while(1)
    {
        /* Sleep for a random period of time */
        sleep(rand() % 10);
        /* generate a random number */
        item = rand() % 100;

        /* wait empty and acquire the mutex lock */
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        /* critical section */
        if(insert_item(item))
            fprintf(stderr, "report error condition\n");
        else
            printf("producer produced %d\n", item);

        /* release the mutex lock and signal full */
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
```

图 4: 生产者线程函数

```c
void *consumer(void *var)
{
    buffer_item item;

    while(1)
    {
        /* sleep for a random period of time */
        sleep(rand() % 10);

        /* wait full and acquire the mutex lock */
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        /* critical section */
        if(remove_item(&item))
            fprintf(stderr, "report error condition\n");
        else
            printf("consumer consumed %d\n", item);

        /* release the mutex lock and signal empty */
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
```

图 5: 消费者线程函数

2.2.3 主函数

1. 接收三个命令行参数：睡眠时间，生产者线程数量，消费者线程数量。其中atoi()能将string转化为int。

```c
/* 1. Get command line arguments argv[1], argv[2], argv[3] */
int sleeptime = atoi(argv[1]);
int producernum = atoi(argv[2]);
int consumernum = atoi(argv[3]);
```

图 6: 获得命令行参数

2. 使用pthread_mutex_init()和sem_init()分别初始化互斥锁与信号量。

```c
/* 2. Initialize mutex lock and semaphore*/
pthread_mutex_init(&mutex, NULL);
sem_init(&full, 0, 0);
sem_init(&empty, 0, BUFFER_SIZE);
```

图 7: 初始化互斥锁与信号量

3. pthread_creat()生成给定数量的生产者、消费者线程。

```c
/* 3. Create the producer threads */
for(int i = 0; i < producernum; i++)
    pthread_create(&tid,NULL,producer,NULL);

/* 4. Create the consumer threads */
for(int i = 0; i < consumernum; i++)
    pthread_create(&tid,NULL,consumer,NULL);
```

图 8: 生成生产者消费者线程

4. 睡眠给定时间后退出。

```c
/* 5. Sleep */
sleep(sleeptime);

/* 6. Exit */
printf("Exiting\n");

exit(0);
```

图 9: 睡眠退出

2.3 测试结果

设置主函数睡眠时间为10，分别生成5个生产者线程，3个消费者线程，测试结果如下：

```
hg@ubuntu:~/Desktop$ gcc -pthread pc.c
hg@ubuntu:~/Desktop$ ./a.out 10 5 3
producer produced 49
consumer consumed 49
producer produced 27
producer produced 59
producer produced 26
producer produced 26
consumer consumed 26
producer produced 11
consumer consumed 11
producer produced 29
consumer consumed 29
producer produced 62
consumer consumed 62
producer produced 35
producer produced 2
Exiting
hg@ubuntu:~/Desktop$
```

图 10: 测试

测试结果正确。

若无消费者，在缓冲满后无法继续插入数据：

```
hg@ubuntu:~/Desktop$ ./a.out 10 5 0
producer produced 35
producer produced 92
producer produced 21
producer produced 27
producer produced 59
Exiting
hg@ubuntu:~/Desktop$
```

图 11: 测试

2.4 项目中出现的问题及解决方案

1. 命令行参数

   当执行程序时，shell会逐一解析命令行参数传递给main函数。具体来说，对于int main(int argc, char *argv[ ], char *envp[ ])，argc表示传给main()的命令行参数个数，*argv[]为一个指向命令行参数的指针数组，每一参数又都是以空字符（null）结尾的字符串，其中argv[0]为程序运行的全路径名称，argv[1]到argv[argc-1]分别为命令行中执行程序名后的第1到argc-1个字符串，argv[argc]为NULL。

   在本试验中需要传递三个参数，在命令行中输入执行程序的命令时，按顺序输入参数：睡眠时间，生产者线程数量，消费者线程数量，在main函数中分别对应使用argv[1]，argv[2]，argv[3]即可。

3 总结及反思

通过这次项目实现我进一步熟悉了如何Pthread API中的信号量与互斥锁保护临界区。