南京信息工程大学 操作系统 实验报告

实验名称 操作系统 日期 2015/5/10 得分 _____ 指导教师 毕硕本

系 理学系 专业 信息与计算科学 班级 1 姓名 王星晨 学号 20152314026

1. 实验目的

- 理解进程同步的信号量机制
- 学会用 C/C++ 实现进程同步的信号量机制
- 学会利用信号量机制解决经典的进程同步问题
- 学会调用操作系统 API 函数创建进程

2. 实验内容

(1) 利用记录型信号量解决生产者 - 消费者问题

(2) 利用 AND 信号量机制解决哲学家进餐问题

(3) 利用记录型信号量解决读者写着问题

3. 实验思路

1. 生产者消费者问题假设生产者和消费者之间的公用缓冲池具有 n 个缓冲区, 可以利用互斥信号量 mutex 实现诸进程对缓冲池的使用; 利用信号量 empty 和 full 分别表示缓冲池中空缓冲区和满缓冲区数量。又假定这些生产者和消费者相互等效，只要缓冲池未满，生产者便可将消息送入缓冲池; 只要缓冲池位空，消费者便可从缓冲池中取走一个消息。定义信号量类型 semaphore 为整型。为了方便, 假设生产的产品为一个随机整数, 则 item 类型定义为整型。程序的变量和类型定义为

```
const int n=5;//缓冲区数量规定为5
int in=0,out=0;
typedef int item;
typedef int semaphore;
semaphore mutex=1,Empty=n,full=0;
item buffer[n];
```

对于 P,V 操作, 定义为

```
void wait (semaphore &S)
{
        while(S<=0);
        S--;
}
void signal (semaphore &S)
{
        S++;
}
```

producer 和 consumer 是两个同时进行的进程, 需要调用操作系统 API 来实现. 为了方便，调用 CreateThread 函数创建 producer 和 consumer 线程来模拟进程.

```
HANDLE handle[2];
handle[0]=CreateThread(NULL,0,producer,NULL,0,NULL);
handle[1]=CreateThread(NULL,0,concumer,NULL,0,NULL);
WaitForMultipleObjects(2,handle,TRUE,-1);
```

2. 哲学家进餐问题

在哲学家进餐问题中, 要求每个哲学家先获得两个临界资源 (筷子) 后方能进餐, 这在本质上是 AND 同步问题。故用 AND 信号量机制可获得最简洁的解法。Swait 和 Ssignal 定义如下

```
  int Swait(semaphore &S1, semaphore &S2)
{
        while (true)
        {
                if (S1 >= 1 && S2 >= 1)
                {
                        S1--;
                        S2--;
                        break;

                }
                else
                {
                        ;
                }

        }
        return 0;
}
int Ssignal(semaphore &S1, semaphore &S2)
{
        S1++;
        S2++;

        return 0;
}
```

3. 读者 - 写者问题

为实现 Reader 与 Writer 进程间在读或写的互斥而设置一个互斥信号量 wmutex. 另外，再设置一个整型变量 readcount 表示在读的进程数目。由于只要有一个 Reader 进程在读，便不允许 Writer 进程去写。因此，仅当 readcount=0, 表示尚无 Reader 进程在读时，Reader 进程才需执行 wait(wmutex) 操作。若 wait(wmutex) 操作成功，Reader 进程便可去读，相应的，做 readcount+1 操作。同理，仅当 Reader 进程在执行了 readcount 减 1 操作后其值为 0 时，才需执行 signal(wmutex) 操作，以便让 Writer 进程写操作。又因为 readcount 是一个可被多个 Reader 进程访问的资源，因此，也应该为它设置一个互斥信号量 mutex.

4. 实验步骤

1. 生产者消费者问题

(1) 打开 Visual Studio , 建立 C++ 项目，添加 ProducerAndConsumer.cpp 文

件

(2) 编写生产者消费者程序代码

(3) 调试运行

2. 哲学家进餐问题

(1) 打开 Visual Studio , 建立 C++ 项目，添加 Philosopher.cpp 文件

(2) 编写哲学家进餐问题程序代码

(3) 调试运行

3. 读者写着问题

(1) 打开 Visual Studio , 建立 C++ 项目，添加 ReadWrite.cpp 文件

(2) 编写读者写者问题程序代码

(3) 调试运行

5. 调试结果

1. 消费者生产者问题

程序在屏幕的部分输出结果

```
producer an item: 41 in buffer 0
concumer an item: 41 in buffer 0
producer an item: 18467 in buffer 1
concumer an item: 18467 in buffer 1
producer an item: 6334 in buffer 2
concumer an item: 6334 in buffer 2
producer an item: 26500 in buffer 3
concumer an item: 26500 in buffer 3
producer an item: 19169 in buffer 4
concumer an item: 19169 in buffer 4
producer an item: 15724 in buffer 0
concumer an item: 15724 in buffer 0
producer an item: 11478 in buffer 1
concumer an item: 11478 in buffer 1
producer an item: 29358 in buffer 2
concumer an item: 29358 in buffer 2
producer an item: 26962 in buffer 3
concumer an item: 26962 in buffer 3
```

producer 不断产生产品 (整数) 放入缓冲区 buffer[i],consumer 不断在缓冲区 buffer 消费产品

2. 哲学家进餐问题

程序在屏幕的部分输出结果

```
philosopher 1 start eating
philosopher 3 start eating
philosopher 1 start thinking
philosopher 5 start eating
philosopher 3 start thinking
```

```
philosopher 2 start eating
philosopher 5 start thinking
philosopher 4 start eating
philosopher 2 start thinking
philosopher 1 start eating
philosopher 4 start thinking
philosopher 3 start eating
philosopher 1 start thinking
philosopher 5 start eating
philosopher 3 start thinking
```

相邻的哲学家是不能同时进餐的，5 个哲学家最多只允许 2 个哲学家同时进餐, 程序结果符合这个特点。

3. 读者写者问题

程序在屏幕的部分输出结果

```
Reading...
Finish reading.
Writing...
Finish writing.
Reading...
Finish reading.
Writing...
Finish writing.
Reading...
Finish reading.
Writing...
Finish writing.
Reading...
Finish reading.
Writing...
Finish writing.
Reading...
```

读和写不能同时进行, 程序结果符合这个特点.

6. 代码附录

1. 生产者消费者问题

ProducerAndConsumer.cpp

```
#include<iostream>
#include <windows.h>
using namespace std;
const int n=5;
int in=0,out=0;
typedef int item;
typedef int semaphore;
semaphore mutex=1,Empty=n,full=0;
item buffer[n];
void wait (semaphore &S)
{
```

```
        while(S<=0);
        S--;
}
void signal (semaphore &S)
{
        S++;
}
DWORD WINAPI producer (LPVOID lpParameter)
{
        do
        {
                item  nextp=rand();
                Sleep(1000);
                wait (Empty);
                wait (mutex);
                buffer[in]=nextp;
                cout<<"producer an item: "<<nextp<<" in
                   buffer "<<in<<endl;
                in=(in+1)%n;
                signal(mutex);
                signal(full);

        }while (TRUE);
        return 0;
}
DWORD WINAPI concumer (LPVOID lpParameter)
{
        do
        {

                wait (full);
                wait (mutex);
                item nextc=buffer[out];
                cout<<"concumer an item: "<<nextc<<" in
                   buffer "<<out<<endl;
                Sleep(2000);
                out=(out+1)%n;
                signal(mutex);
                signal(Empty);

        }while (TRUE);
        return 0;
}
int main()
{
        HANDLE handle[2];
        handle[0]=CreateThread(NULL,0,producer,NULL,0,
           NULL);
```

```
        handle[1]=CreateThread(NULL,0,concumer,NULL,0,
           NULL);
        WaitForMultipleObjects(2,handle,TRUE,-1);
        return 0;
}
```

2. 哲学家进餐问题

Philosopher.cpp

```
#include<iostream>
#include<Windows.h>
using namespace std;
typedef int semaphore;
semaphore chopstick[5] = { 1,1,1,1,1 };
int Swait(semaphore &S1, semaphore &S2)
{
        while (true)
        {
                if (S1 >= 1 && S2 >= 1)
                {
                        S1--;
                        S2--;
                        break;

                }
                else
                {
                        ;
                }

        }
        return 0;
}
int Ssignal(semaphore &S1, semaphore &S2)
{
        S1++;
        S2++;

        return 0;
}
DWORD WINAPI P(LPVOID lpParameter)
{
        int *i = (int *)(lpParameter);
        do {

                Swait(chopstick[(*i + 1) % 5], chopstick
                   [*i]);
                cout << "philosopher " << *i + 1 << "
                   start eating\n";
                Sleep(2000);            //eating
```

```
                cout << "philosopher " << *i + 1 << "
                   start thinking\n";
                Sleep(100);//put down the chopsticks
                Ssignal(chopstick[(*i + 1) % 5],
                   chopstick[*i]);
                Sleep(3000);            //thinking
        } while (true);
}

int main()
{
        int philosopher[5] = { 0,1,2,3,4 };
        HANDLE handle[5];
        for (int i = 0; i < 5; i++)
        {
                handle[i] = CreateThread(NULL, 0, P, &
                   philosopher[i], 0, NULL);
                Sleep(50);
        }
        WaitForMultipleObjects(5, handle, 1, -1);
        return 0;
}
```

3. 读者写者问题

ReadWrite.cpp

```
#include<iostream>
#include<windows.h>
using namespace std;
typedef int semaphore;
semaphore rmutex=1,wmutex=1;
int readcount=0;
int signal(semaphore &S)
{
        S++;
        return 0;
}
int wait(semaphore &S)
{
        while(S<=0);
        S--;
        return 0;
}
DWORD WINAPI Reader(LPVOID lpParameter)
{
        do{
                wait(rmutex);
                if(readcount==0)wait(wmutex);
                readcount++;
                signal(rmutex);
```

```
                cout<<"Reading...\n";
                Sleep(1000);
                wait(rmutex);
                readcount--;
                cout << "Finish reading.\n";
                if(readcount==0)
                        signal(wmutex);
                signal(rmutex);

        }while(true);
        return 0;
}

DWORD WINAPI Writer(LPVOID lpParameter)
{
        do{
                wait(wmutex);
                cout<<"Writing...\n";
                Sleep(3000);
                cout << "Finish writing.\n";
                signal(wmutex);

        }while(true);

}
int main()
{
        HANDLE handle[2];
        handle[0]=CreateThread(NULL,0,Reader,NULL,0,NULL)
           ;
        handle[1]=CreateThread(NULL,0,Writer,NULL,0,NULL)
           ;
        WaitForMultipleObjects(2,handle,true,-1);
        return 0;
}
```