“读者-写者问题”

利用记录型**信号量**分别给出
“读写无优先、写者优先”
问题的同步算法

第七小组

汇报人：胡才郁，张俊雄

小组成员：胡峻豪，胡天磊

记录型信号量

用户进程可以通过使用操作系统提供的一对**原语**来对**信号量**进行操作，从而很方便的实现了进程互斥、进程同步。

原语是一种特殊的程序段，其执行只能**一气呵成，不可被中断**。

```
struct {
    int value; // 代表资源数目
    struct process *L; // 链接所有等待该资源的进程
} semaphore;
```

相当于申请资源

当S.value < 0时，表示该资源已经分配完毕

需要调用block()原语使进程从运行态变为阻塞态，并把当前进程挂到信号量S的等待队列当中

P操作

```
void wait(semaphore S) {
    S.value --;
    if (S.value < 0) {
        block(S.L);
    }
}
```

相当于释放资源

当S.value <= 0时，表示当前仍有等待该资源的进程被阻塞在信号量S的等待队列当中

需要调用wakeup()原语将S.L中的第一个进程从阻塞态变为就绪态

V操作

```
void signal(semaphore S) {
    S.value ++;
    if (S.value <= 0) {
        wake(process p);
    }
}
```

使用信号量实现互斥和同步

互斥

```
semaphore mutex = 1;

P1() {
    p(mutex);
    访问临界资源;
    v(mutex);
}
```

1. 设置互斥信号量mutex，并设初值为1。
2. 在访问临界资源之前，执行p(mutex)，把mutex的值-1，表示当前有进程正在访问临界资源。
3. 在结束访问之后，执行v(mutex)，把mutex的值+1，表示访问临界资源的进程访问结束。

同步

```
semaphore s = 0;

p1() {
    x;
    v(s);
}

p2() {
    p(s);
    y;
}
```

1. 设置同步信号量s，初值为0。
2. 在先要执行的代码之后执行v操作。
3. 在后执行的代码之前执行p操作。

基本要求

1. 允许多个读者进程同时对文件执行读操作
2. 某一时刻只允许一个写者进程向文件中写信息
3. 任一写着在完成写操作之前不允许其他读者或写者工作
4. 写者执行写操作前直到已有的读者和写者全部退出

互斥关系

1. 读进程与写进程之间　　互斥
2. 写进程与写进程之间　　互斥
3. 读进程与读进程之间　　不互斥

实现方式

1. 设置整数变量`count`记录当前有几个读进程在访问文件
2. 第一个读进程　　加锁
3. 最后一个读进程　　解锁

信号量、控制变量设置

```
semaphore rw = 1; // 用于实现对文件的互斥访问 表示当前是否有进程访问文件
int readcount = 0; // 记录当前有几个读进程在访问文件
```

互斥关系

1. 读进程与写进程之间　　互斥
2. 写进程与写进程之间　　互斥
3. 读进程与写进程之间　　不互斥

写进程

```
writer(){
  while(1){
    P(rw);    // 写之前加锁
    写文件....
    V(rw);    // 写之后解锁
  }
}
```

读进程（读进程阻塞）

```
reader(){
  while(1){
    if(readcount == 0)
      P(rw);  // 第一个读进程负责加锁
    readcount++; // 读者在读数+1

    读文件...

    readcount--; // 读者在读数-1
    if(readcount == 0)
        V(rw);  // 最后一个一个读进程负责解锁
  }
}
```

弊端

若两个读进程并发执行
则两个读进程有可能先后执行
P(rw)
从而使得第二个读进程出现阻塞

信号量、控制变量设置

```
semaphore rw = 1; // 用于实现对文件的互斥访问 表示当前是否有进程访问文件
int readcount = 0; // 记录当前有几个读进程在访问文件
semaphore reader_mutex = 1; // 用于保证对readcount变量的互斥访问
```

互斥关系

1. 读进程与写进程之间　　互斥
2. 写进程与写进程之间　　互斥
3. 读进程与写进程之间　　不互斥

写进程

```
writer(){
  while(1){
    P(rw);    // 写之前加锁
    写文件....
    V(rw);    // 写之后解锁
  }
}
```

读进程

```
reader(){
  while(1){
    P(reader_mutex); // 各读进程互斥访问reader_count
    if(readcount == 0)
        P(rw);  // 第一个读进程负责加锁
    readcount++; // 读者在读数+1
    V(reader_mutex);

    读文件...

    P(reader_mutex); // 各读进程互斥访问count
    readcount--; // 读者在读数-1
    if(readcount == 0)
        V(rw);  // 最后一个一个读进程负责解锁
    V(reader_mutex);
  }
}
```

解决方法

设置mutex互斥信号量
保证各个读进程对于count访问互斥
此时count成为临界资源

读者优先

只要读进程还在读，
写进程就一直阻塞等待，
写进程“饥饿”

信号量、控制变量设置

```
semaphore rw = 1; // 用于实现对文件的互斥访问 表示当前是否有进程访问文件
int readcount = 0; // 记录当前有几个读进程在访问文件
semaphore reader_mutex = 1; // 用于保证对readcount变量的互斥访问
```

写进程

```
writer(){
  while(1){
    P(rw);    // 写之前加锁
    写文件....
    V(rw);    // 写之后解锁
  }
}
```

读进程

```
reader(){
  while(1){
    P(reader_mutex); // 各读进程互斥访问reader_count
    if(readcount == 0)
        P(rw);  // 第一个读进程负责加锁
    readcount++; // 读者在读数+1
    V(reader_mutex);

    读文件...

    P(reader_mutex); // 各读进程互斥访问count
    readcount--; // 读者在读数-1
    if(readcount == 0)
        V(rw);  // 最后一个一个读进程负责解锁
    V(reader_mutex);
  }
}
```

解决方法

设置mutex互斥信号量
保证各个读进程对于count访问互斥
此时count成为临界资源

写者优先

当写者到来时，队列中此写者之前的读者仍按照顺序继续读操作，但之后到来的读进程都阻塞，直到队列中不存在写进程在排队。

```
semaphore rw = 1; // 用于实现对文件的互斥访问 表示当前是否有进程访问文件
semaphore read = 1; // 写进程优先互斥
int readcount = 0; // 记录当前有几个读进程在访问文件
int writercount = 0; // 记录当前有几个写进程正在排队
semaphore reader_mutex = 1; // 用于保证对readcount变量的互斥访问
semaphore writer_mutex = 1; // 用于保证对writercount变量的互斥访问
```

假设：1号写者(正在读)-1号读者-2号写者-2号读者-3号写者

```
writer(){
  while(1){
    P(writer_mutex); // 各写进程互斥访问writercount
    if(writercount != 0) // 如果当前写者队列不为空则申请读者优先
      P(read);
    writecount++; // 写者数量+1
    V(writer_mutex);

    P(rw);
    写文件....
    V(rw);

    P(writer_mutex); // 各写进程互斥访问writercount
    writecount--; // 写者数量-1
    if(writecount != 0) // 如果写者队列为空则允许读者进行操作
      V(read);
    V(writer_mutex); // 各写进程互斥访问writercount
  }
}
```

```
reader(){
    while(1){
        P(read);  // 申请读取权限

        P(reader_mutex); // 各读进程互斥访问reader_count
        if(readcount == 0)
            P(rw);  // 第一个读进程负责加锁
        readcount++; // 读者在读数+1
        V(reader_mutex);

        读文件...

        P(reader_mutex); // 各读进程互斥访问count
        readcount--; // 读者在读数-1
        if(readcount == 0)
            V(rw);  // 最后一个一个读进程负责解锁
        V(reader_mutex);
    }
}
```

读写公平

```
semaphore rw = 1;//实现对共享文件的互斥访问，读和写
int count = 0; //记录当前有几个进程在访问文件
semaphore mutex = 1; //实现对count变量的互斥访问
semaphore w = 1; //实现读写公平
```

```
writer() {
    while(1) {
        p(w);
        p(rw); //写之前加锁
        写文件....
        v(rw); //写之前解锁
        v(w);
    }
}
```

```
reader() {
    while(1) {
        p(w);
        p(mutex); //对count变量的检查和赋值无法一气呵成加锁
        if (count == 0) {
            p(rw); //当前没有读进程在访问，对读进程加锁
        }
        count++; //当前正在访问的读进程数+1
        v(mutex);
        v(w);
        读文件....
        p(mutex);
        count--; //当前正在访问的读进程数-1
        if (count == 0) {
            v(rw); //当前为最后一个读进程，对读进程解锁
        }
        v(mutex);
    }
}
```

三种情况：

- 写者 写者
- 写者 读者
- 读者 读者
- 读者 写者

1. 写者先执行时，其余写者和读者就不可能执行。

2. 读者先执行时，另一个读者也可执行，写者不可能执行。

谢谢观看

“读者-写者”问题

第七小组

汇报人：胡才郁，张俊雄

小组成员：胡峻豪，胡天磊