|  |
| --- |
| **국민대학교 컴퓨터공학부** |
| Operating System |
| **Homework #2 Semaphore & Condition Variable** |

|  |
| --- |
| 20093267 김 성 근  Due Date. 2014-04-07 |

**1. Monitor (a lock & Condition variables)로 Semaphore를 구현하시오.**

**class Semaphore implement Monitor{**

**int value // Only positive integer value**

**Condition semaCondition**

**Sempaphor( int initialValue ){ // This is a constructor**

**value = initialValue**

**}**

**Lock lock(FREE) // Lock value is initially FREE**

**P( ){**

**lock->Acquire( )**

**while ( value == 0 ) { // Non negative integer**

**semaCondition.wait(&lock); // An atomic operation that waits for**

**// Semaphore to become positive,**

**}**

**decrement value by 1**

**lock ->Release( )**

**}**

**V( ){**

**lock->Acquire( )**

**increment value by 1**

**semaCondition.signal( ) // waking up a waiting P, if any**

**lock ->Release( )**

**}**

**}**

**class Condition{**

**Queue queue = NIL;**

**Wait(lock){**

**release lock, go to sleep (atomic operation)**

**re-acquire lock**

**}**

**Signal(){**

**wake up a waiter**

**}**

**BroadCast(){ // Don’t use in Semaphore**

**wake up all waiter**

**}**

**}**

**Semaphore Condition**

1. Have a positive integer value (No negative values)

위 조건에 의해 **[ int value ]** 가 필요하다.

2. Have two operations

- P() : an atomic operation that waits for semaphore to become positive,

then decrements it by 1

- V() : an atomic operation that increments semaphore by 1,

waking up a waiting P, if any

위 조건에 의해 [ **P( ), V( )** ] 가 구현되며 하는 일도 위와 똑같다.

3. 기타 조건 :

- Only operation are P and V – can’t read or write value, except to set it initially

- Operation must be atomic

**Monitor**

- Have a lock (Lock is initially FREE )

위 조건에 의해 **[ Lock lock(FREE) ]**이 구현되고 초기값은 FREE 이다.

- Have zero or more condition variable for managing concurrent access to shared data

위 조건에 의해 Condition이 구현되며, 이 Condition은 P( )가 실행될 때 value 가 0 이면 sleep이 되는 곳이다.

**Condition Variable**

1. A queue of threads waiting for something inside a critical section

2. Have three operations

- Wait() : release lock, go to sleep, re-acquire lock,

Releasing lock and going to sleep is atomic

- Signal() : wake up a waiter, if any

- Broadcast() : wake up all waiters

3. Rule : Must hold lock when doing condition variable operations

위의 조건에 의해서 Condition Variable이 구현되고 이것을 Semaphore에서 사용된다.

**2. Semaphore로 Condition Variable을 구현 하시오. (Extra credit)**

**- Semaphore 의 int value를 직접적으로 확인하는 상태의 해결방법**

**class Condition{**

**Semaphore semaphore // Condition Variable의 Queue를 저장하는 공간**

**BinarySemaphore lockSemaphore // Wait에서 받아오는 Lock 이외에 Mutal Exclusion 하기 위해**

**// 사용된 Semaphore ( Binary Semaphore는 lock으로 이용 가능)**

**int waiters = 0; // Queue를 이용하는 Semaphore의 상태를 확인 할 수 없음으로 밖에 Count 할 수 있는**

**// 변수를 두어 Wait( )가 실행되는지를 확인한다.**

**Wait( Lock \*lock ){**

**lockSemaphore ->P( )**

**waiters++**

**lockSemaphore ->V( )**

**lock->Release( )**

**semaphore->P( )**

**lock->Acquire( )**

**}**

**Singal( ){**

**lockSemaphore -> P( )**

**if(waiters > 0){**

**waiters--**

**semaphore->V( )**

**}**

**lockSemaphore -> V( )**

**}**

**BroadCast( ){**

**lockSemaphore -> P( )**

**while(waiters > 0 ){**

**waiters--;**

**semaphore->V( )**

**}**

**lockSemaphore -> V( )**

**}**

**}**

■ Semaphore 상태를 직접적으로 확인할 수 없어서 그것을 해결하기 위해 추가된 부분