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

Definition of Semaphore

- A kind of generalized lock

- Main synchronization primitive use in UNIX

Condition

- Have a positive integer value

- 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

- No negative values

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

- Operation must be atomic

Definition of Monitor

- Have a lock

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

Lock

- Provides mutual exclusion to the shared data.

- Lock::Acquire : wait until lock is free, then grab it

- Lock::Release : unlock, wake up anyone waiting in Acquire

Rules of using a lock

- Always acquire before accessing shared data structure

- Always release after finishing with shared data

- Lock is initially FREE

Condition Variable

- A queue of threads waiting for something inside a critical section

- 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

Rule

-Must hold lock when doing condition variable operations

class Semaphore implement Monitor{

Integer value; // Only positive integer value

Condition semaCondition

Sempaphor( initialValue ){ // This is constructor

value = initialValue

}

Lock lock

lock.value = 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(){ // not use in Semaphore

wake up all waiter

}

}

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

1.

class Condition{

Semaphore semaphore // Role is save queue

BinarySemaphore lockSemaphore // Role is Lock

int waiters = 0;

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()

}

}

class Semaphore(){

int value = value;

Queue queue = NIL;

P(){

}

V(){

}

}

2.여기서 추가 되는 것 waiter를 증가시키지 않음으로써 먼저 들어 왔던 것을 처리하게 되는 작업

class Condition{

Semaphore semaphore // Role is save queue

Semaphore latestSemaphore

BinarySemaphore lockSemaphore // Role is Lock

int waiters = 0;

Wait(Lock \*lock){

lockSemaphore ->P()

waiters++

lockSemaphore ->V()

lock->Release()

semaphore->P()

latestSemaphore->V()

lock->Acquire()

}

Singal(){

lockSemaphore -> P()

if(waiters > 0){

waiters--;

semaphore->V()

latestSemaphore->P()

}

lockSemaphore -> V()

}

BroadCast(){

lockSemaphore -> P()

for(i=0; i<waiters; i++)

semaphore->V()

while(waiters > 0 ){

waiters--;

latestSemaphore->P()

}

lockSemaphore -> V()

}

}