ROVIN 2010 Robot Vision & Intelligence Lab.

고급프로그래밍

멀티스레딩

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), **클래스와 객체 (4)**	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		**중간고사**		
9		파일 입출력(14), 예외처리 및 C 사용(15)		6
10		람다식(16, 5/7) , 멀티스레딩(17, 5/9)	6	7
11		멀티스레딩(18, 5/14), 고급문법		
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		**기말고사**		

오늘의 학습내용

- 멀티스레딩 - 계속

Deadlock

- 교착상태
  - 복수의 스레드 작업이 있을 때, 서로 무한 대기 상태에 이름

thread 1

```
void thread1_fun( )
{
    unique_lock<mutex> lock1(_mutex1);
    unique_lock<mutex> lock2(_mutex2);

    cout << "Locked! << endl;
}
```

← 무한 대기 →

thread 2

```
void thread2_fun( )
{
    unique_lock<mutex> lock2(_mutex2);
    unique_lock<mutex> lock1(_mutex1);

    cout << "Locked! << endl;
}
```

Deadlock

- 교착상태
  - 복수의 스레드 작업이 있을 때, 서로 무한 대기 상태에 이름

thread 1

```
void thread1_fun( )
{
    lock_guard<mutex> lock1(_mutex1);
    lock_guard <mutex> lock2(_mutex2);

    cout << "Locked! << endl;
}
```

← 무한 대기 →

thread 2

```
void thread2_fun( )
{
    lock_guard <mutex> lock2(_mutex2);
    lock_guard <mutex> lock1(_mutex1);

     cout << "Locked! << endl;
}
```

Deadlock

- 교착상태
  - 복수의 스레드 작업이 있을 때, 서로 무한 대기 상태에 이름

thread 1

```
void thread1_fun( )
{
    lock_guard<mutex>  lock1(_mutex1);
    lock_guard <mutex> lock2(_mutex2);

    cout << "Locked! << endl;
}
```

교착상태 회피 →

thread 2

```
void thread2_fun( )
{
   while(true)
   {
      _mutex2.lock();
      if(!_mutex1.try_lock())
      {
          _mutex2.unlock();
          continue;
      }

      cout << "Locked! << endl;
      _mutex1.unlock();
      _mutex2.unlock();
}
```

Deadlock

- 교착상태
  - 복수의 스레드 작업이 있을 때, 서로 무한 대기 상태에 이름

thread 1

```
void thread1_fun( )
{
    lock_guard<mutex>  lock1(_mutex1);
    lock_guard <mutex> lock2(_mutex2);

    cout << "Locked! << endl;
}
```

thread 2

```
void thread2_fun( )
{
   while(true)
   {
      unique_lock<mutex> lock2(_mutex2);
      unique_lock<mutex> lock1(_mutex1, try_to_lock);

      if (lock1.owns_lock() == true) {
            cout << "Locked! << endl;
            break;
       }
    }
}
```

교착상태 회피 →

Deadlock

- 교착상태
  - 복수의 스레드 작업이 있을 때, 서로 무한 대기 상태에 이름

thread 1

```
void thread1_fun( )
{
    unique_lock<mutex> lock1(_mutex1, defer_lock);
    unique_lock<mutex> lock2(_mutex2, defer_lock);

    lock ( lock1, lock2 );
    cout << "Locked! << endl;
}
```

thread 2

```
void thread2_fun( )
{
    unique_lock<mutex> lock1(_mutex1, defer_lock);
    unique_lock<mutex> lock2(_mutex2, defer_lock);

    lock ( lock1, lock2 );
    cout << "Locked! << endl;
}
```

교착상태 회피

Condition Variable

- 멀티스레드 사이의 동기화를 안정적으로 구현하는데 필요
- 특정 조건이 만족되기 전까지 기다리다가, 조건이 만족되면 해당 스레드를 깨우는 데 사용
- 일반적으로 lock 변수와 함께 사용

Main Thread

```
queue<string>     mQueue;
mutex             mMutex;
condition_variable mCondVar;
   :

unique_lock<mutex> lock1(mMutex);
mQueue.push( --- );
mCondVar.notify_all();
```

Background Thread

```
unique_lock<mutex> lock2(mMutex);
while(true)
{
   mCondVar.wait(lock2);

   //알림이 수신되면 처리
    :
}
```

Conditional Variable

- 예제

```
#include <iostream>
#include <condition_variable>
#include <mutex>
#include <thread>

mutex               mutex_;
condition_variable  condVar;

void doTheWork(){
   cout << "Processing shared data." << endl;
}
void waitingForWork(){
   cout << "Worker: Waiting for work." << endl;
   unique_lock<std::mutex> lock(mutex_);
   condVar.wait(lock);
   doTheWork();
   cout << "Work done." << std::endl;
}
void setDataReady(){
   cout << "Sender: Data is ready." << endl;
   condVar.notify_one();
}
```

```
int main()
{
   cout << endl;
   thread t1(waitingForWork);
   thread t2(setDataReady);

   t1.join();
   t2.join();
   cout << endl;
}
```

```
Worker: Waiting for work //Receiver가 기다리다가
Sender: Data is ready.   //Sender가 준비되면
Processing shared data.  //Notification을 전달하고
Work done.               //Receiver가 작업을 완료
```

Conditional Variable

- 예제

```
:
#include <condition_variable>
:
mutex m;
condition_variable cv;
string data;
bool ready = false, processed = false;

void worker_thread()
{
   // Wait until main() sends data
   unique_lock lk(m);
   cv.wait(lk, []{ return ready; });
   // after the wait, we own the lock.
   cout << "Worker thread is processing data\n";
   data += " after processing";
   // Send data back to main()
   processed = true;
   cout << "Worker thread signals data processing completed\n";
   //notify 이전에 unlock 수행
   lk.unlock();
   cv.notify_one();
}
```

```
while(!predicate())
   wait(lk)

wait(unique_lock<mutex>& lk, predicate)
 → notify 되어도 predicate이 false이면 계속 기다림
```

Conditional Variable

- 예제

```
int main()
{
    thread th(worker_thread);

    data = "Example data";
    // send data to the worker thread
   {
       lock_guard lk(m);
       ready = true;
       cout << "main() signals data ready for processing₩n";
   }
   cv.notify_one();

   // wait for the worker
   {
      unique_lock lk(m);
      cv.wait(lk, []{ return processed; });
   }
   cout << "Back in main(), data = " << data << '₩n';

   th.join();
}
```

Conditional Variable

- notify_all : 대기중인 모든 스레드에게 알림
- wait_for : wait 함수와 동일하며 시간제한 추가

```
using namespace std::chrono_literals
```

```
wait_for(lk, 100ms, []{ return true; });
```

예제 1

- Logger Class
  - 다른 스레드로부터 메시지를 받아 큐에 저장
  - 백그라운드 스레드로 큐에 들어오는 메시지를 파일에 저장

예제 1

```
#include <queue>
#include <string>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

class Logger {
public:
   //Starts a background thread writing log entries to a file.
    Logger(); // Add log entry to the queue.
    void log(const std::string& entry);
protected:
    void processEntries();
    mutex                 mMutex;
    condition_variable  mCondVar;
    queue<std::string>   mQueue;
    thread mThread; // The background thread.

private: // Prevent copy construction and assignment.
   Logger(const Logger& src);
   Logger& operator=(const Logger& rhs);
};
```

```
Logger::Logger() {
    // Start background thread.
    mThread = thread{&Logger::processEntries, this};
}
void Logger::log(const std::string& entry) {
   unique_lock<mutex> lock(mMutex);
    mQueue.push(entry);
    mCondVar.notify_all(); // Notify condition variable to wake up thread.
}
void Logger::processEntries()
{
   ofstream ofs("log.txt"); // Open log file.
   if (ofs.fail()) {
       cerr << "Failed to open logfile." << endl;  return;
   }

   // Start processing loop.
   unique_lock<mutex> lock(mMutex);
   while (true) {
     mCondVar.wait(lock); // Wait for a notification.

    // Condition variable is notified, so something is in the queue.
     lock.unlock();
     while (true) {
           lock.lock();
           if (mQueue.empty())  break;
           else {
                ofs << mQueue.front() << endl;
                mQueue.pop();
            }
            lock.unlock();
     }
 } }
```

예제 1

- main.cpp

```cpp
#include "Logger.h"
#include <iostream>
#include <sstream>
#include <thread>
#include <vector>

using namespace std;

void logSomeMessages(int id, Logger& logger)
{
    for (int i = 0; i < 10; ++i) {
        stringstream ss;
        ss << "Log entry " << i << " from thread " << id;
        logger.log(ss.str());
    }
}

int main()
{
    Logger logger;

    vector<thread> threads;
    // Create a few threads all working with the same Logger instance.
    for (int i = 0; i < 10; ++i) {
        threads.push_back(thread{logSomeMessages, i, ref(logger)});
    }

    // Wait for all threads to finish.
    for (auto& t : threads) {
        t.join();
    }

    return 0;
}
```

예제 1의 문제점

- Logger 객체의 백그라운드 스레드 종료 전 소멸(비정상 종료)

```cpp
#include "Logger.h"
#include <iostream>
#include <sstream>
#include <thread>
#include <vector>

using namespace std;

void logSomeMessages(int id, Logger& logger)
{
    for (int i = 0; i < 10; ++i) {
        stringstream ss;
        ss << "Log entry " << i << " from thread " << id;
        logger.log(ss.str());
    }
}

int main()
{
    Logger logger;

    vector<thread> threads;
    // Create a few threads all working with the same Logger instance.
    for (int i = 0; i < 10; ++i) {
        threads.push_back(thread{logSomeMessages, i, ref(logger)});
    }

    // Wait for all threads to finish.
    for (auto& t : threads) {
        t.join();
    }

    return 0;
}
```

예제 2

```
#include <queue>
#include <string>
#include <thread>
#include <mutex>
#include <condition_variable>

class Logger
{
public:
    // Starts a background thread writing log entrie
    Logger();
    // Gracefully shut down background thread.
    virtual ~Logger();
    // Add log entry to the queue.
    void log(const std::string& entry);

protected:
    void processEntries();
    bool mExit;
    // Mutex and condition variable to protect acces
    std::mutex mMutex;
    std::condition_variable mCondVar;
    std::queue<std::string> mQueue;
    // The background thread.
    std::thread mThread;

private:
    // Prevent copy construction and assignment.
    Logger(const Logger& src);
    Logger& operator=(const Logger& rhs);
};
```

추가 ----→

예제 2

```cpp
#include <fstream>
#include <iostream>

using namespace std;

Logger::Logger() : mExit(false)
{
    // Start background thread.
    mThread = thread{&Logger::processEntries, this};
}

Logger::~Logger()
{
    // Gracefully shut down the thread by setting mExi
    // to true and notifying the thread.
    mExit = true;
    // Notify condition variable to wake up thread.
    mCondVar.notify_all();
    // Wait until thread is shut down.
    mThread.join();
}
```

```cpp
void Logger::processEntries()
{
    // Open log file.
    ofstream ofs("log.txt");
    if (ofs.fail()) {
        cerr << "Failed to open logfile." << endl;
        return;
    }

    // Start processing loop.
    unique_lock<mutex> lock(mMutex);
    while (true) {
        // Wait for a notification.
        mCondVar.wait(lock);

        // Condition variable is notified, so some
        // and/or we need to shut down this thread
        lock.unlock();
        while (true) {
            lock.lock();
            if (mQueue.empty()) {
                break;
            } else {
                ofs << mQueue.front() << endl;
                mQueue.pop();
            }
            lock.unlock();
        }
        if (mExit)
            break;
    }
}
```

예제 2 문제점

1. mThread의 알림을 기다리기 전에 Logger 소멸자에서 notify_all() 호출
2. mThread에서 이 알림을 놓침
3. mThread.join() 과 mCondVar.wait()에서 무한 블로킹(deadlock)

예제 3

```
class Logger
{
public:
    // Starts a background thread writing log entries to a file.
    Logger();
    // Gracefully shut down background thread.
    virtual ~Logger();
    // Add log entry to the queue.
    void log(const std::string& entry);

protected:
    void processEntries();
    bool mThreadStarted;
    bool mExit;
    // Mutex and condition variable to protect access to the queue.
    std::mutex mMutex;
    std::condition_variable mCondVar;
    std::queue<std::string> mQueue;
    // The background thread.
    std::thread mThread;

    // Mutex and condition variable to detect when the thread
    // starts executing its loop.
    std::mutex mMutexStarted;
    std::condition_variable mCondVarStarted;

private:
    // Prevent copy construction and assignment.
    Logger(const Logger& src);
    Logger& operator=(const Logger& rhs);
};
```

예제 3

```
Logger::Logger() : mThreadStarted(false), mExit(false)
{
    // Start background thread.
    mThread = thread{&Logger::processEntries, this};
    // Wait until background thread starts its processing loop.
    unique_lock<mutex> lock(mMutexStarted);
    mCondVarStarted.wait(lock, [&](){return mThreadStarted == true;});
}

Logger::~Logger()
{
    // Gracefully shut down the thread by setting mExit
    // to true and notifying the thread.
    mExit = true;
    // Notify condition variable to wake up thread.
    mCondVar.notify_all();
    // Wait until thread is shut down.
    mThread.join();
}

void Logger::log(const std::string& entry)
{
    // Lock mutex and add entry to the queue.
    unique_lock<mutex> lock(mMutex);
    mQueue.push(entry);

    // Notify condition variable to wake up thread.
    mCondVar.notify_all();
}
```

```
void Logger::processEntries()
{
    // Open log file.
    ofstream ofs("log.txt");
    if (ofs.fail()) {
        cerr << "Failed to open logfile." << endl;
        return;
    }

    // Start processing loop.
    unique_lock<mutex> lock(mMutex);

    // Notify listeners that thread is starting pro
    mThreadStarted = true;
    mCondVarStarted.notify_all();

    while (true) {
        // Wait for a notification.
        mCondVar.wait(lock);

        // Condition variable is notified, so somet
        // and/or we need to shut down this thread.
        lock.unlock();
        while (true) {
            lock.lock();
            if (mQueue.empty()) {
                break;
            } else {
                ofs << mQueue.front() << endl;
                mQueue.pop();
            }
            lock.unlock();
        }
        if (mExit)
            break;
    }
}
```

while(!predicate())
wait(lk)

예제 3

```
Logger::Logger() : mThreadStarted(false), mExit(false)
{
    // Start background thread.
    mThread = thread{&Logger::processEntries, this};
    // Wait until background thread starts its processing loop.
    unique_lock<mutex> lock(mMutexStarted);
    mCondVarStarted.wait(lock, [&](){return mThreadStarted == true;});
}

Logger::~Logger()
{
    // Gracefully shut down the thread by setting mExit
    // to true and notifying the thread.
    mExit = true;
    // Notify condition variable to wake up thread.
    mCondVar.notify_all();
    // Wait until thread is shut down.
    mThread.join();
}

void Logger::log(const std::string& entry)
{
    // Lock mutex and add entry to the queue.
    unique_lock<mutex> lock(mMutex);
    mQueue.push(entry);

    // Notify condition variable to wake up thread.
    mCondVar.notify_all();
}
```

'=' 해당함수의 모든 변수를 사용
'&' 모든 변수를 참조형으로 사용
'&a' 특정변수만 참조형 사용

```
void Logger::processEntries()
{
    // Open log file.
    ofstream ofs("log.txt");
    if (ofs.fail()) {
        cerr << "Failed to open logfile." << endl;
        return;
    }

    // Start processing loop.
    unique_lock<mutex> lock(mMutex);

    // Notify listeners that thread is starting pro
    mThreadStarted = true;
    mCondVarStarted.notify_all();

    while (true) {
        // Wait for a notification.
        mCondVar.wait(lock);

        // Condition variable is notified, so somet
        // and/or we need to shut down this thread.
        lock.unlock();
        while (true) {
            lock.lock();
            if (mQueue.empty()) {
                break;
            } else {
                ofs << mQueue.front() << endl;
                mQueue.pop();
            }
            lock.unlock();
        }
        if (mExit)
            break;
    }
}
```