Chapter 7. 멀티 스레드 메모리 풀링

- 여러 스레드가 메모리를 동시에 할당하고 해지 하기 위해서, 상호 배제를 할당기 메소드에 추가해야 함.

**버전 4. 구현**

- 멀티 스레드 메모리 풀을 구현

|  |
| --- |
| template<class T>  class MemoryPool  {  public:  MemoryPool(size\_t size = EXPANSION\_SIZE)  {  expandTheFreeList();  }  ~MemoryPool()  {  MemoryPool<T> \*nextptr = next;  for (nextprt = next; nextptr != nullptr; nextptr = next)  {  next = next->next;  delete[] nextptr;  }  }  // 자유 리스트로부터 T요소를 할당  inline void\* alloc(size\_t size)  {  if (!next)  expandTheFreeList();  MemoryPool<T> \*head = next;  next = head->next;  return head;  }  // 자유 리스트로 T요소를 반환  inline void\* free(void \*someElement)  {  MemoryPool<T> \*head = static\_cast<MemoryPool<T>\*>(someElement);  head->next = next;  next = head;  }  private:  // 자유 요소를 자유리스트에 추가  static void expandTheFreeList()  {  // 다음 포인터를 포함할 수 있도록 객체를 크게 할당해야 한다.  size\_t size = (sizeof(T) > sizeof(MemoryPool<T>\*)) ? sizeof(T) : sizeof(MemoryPool<T>\*);  MemoryPool<T> \* runner = static\_cast<MemoryPool<T>\*> (new char[size]);  next = runner;  for (int i = 0; i < EXPANSION\_SIZE; ++i)  {  runner->next = static\_cast<MemoryPool<T>\*> (new char[size]);  runner = runner->next;  }  runner->next == nullptr;  }  // 자유 리스트가 비었다면 이 양만큼 확장  enum { EXPANSION\_SIZE = 32 };  // 다음 요소  MemoryPool<T> \*next;  }; |
| template <class POOLTYPE, class LOCK>  class MTMemoryPool  {  public:  // freeList로 부터 요소를 할당받는다.  inline void\* alloc(size\_t size)  {  void\* mem;  theLock.lock();  mem = stPool.alloc(size);  theLock.unlock();  return mem;  }  // 요소를 freeList에 반환한다.  inline void free(void\* someElement)  {  theLock.lock();  stPool.free(someElement);  theLock.unlock();  }  private:  POOLTYPE stPool; // 단일 스레드 풀, 6장 버전2. 메모리풀을 사용  LOCK theLock;  }; |

- alloc 메소드는 할당을 메모리 풀 멤버로 일임하고, 잠금을 잠금 멤버로 일임

- MTMemoryPool 템플릿의 인스턴스를 만들기 위해 메모리 풀 형식과 잠금 형식을 제공해야 한다.

|  |
| --- |
| // 추상 기본 클래스  class Lock  {  public:  virtual ~Lock() {}  virtual void lock() = 0;  virtual void unlock() = 0;  };  class MutexLock : public Lock  {  public:  inline void lock() { mutex.lock(); }  inline void unlock() { mutex.unlock(); }  private:  std::mutex mutex;  }; |

- 멀티스레드 메모리 풀의 인스턴스를 만들기 위해 Rational의 구현**(6장 버전 2.)**을 수정

- 메모리 풀은 MemoryPool<Rational>형식, MutexLock이 잠금 역할을 수행

**키 포인트**

- 전역 메모리 관리자 (new()와 delete()가 구현하는)는 일반적인 목적을 가지고 있기 때문에 부하를 많이 준다.

- 대부분 단일 스레드에 한정된 메모리 블록을 할당한다면 비약적인 성능 개선을 꾀할 수 있다. 단일 스레드 메모리 관리자는 멀티 스레드 메모리 관리자 보다 훨씬 빠르다.

- 효율적인 단일 스레드 할당기를 여러 개 개발한다면, 템플릿을 사용하여 이 할당기들을 멀티스레드 환경으로 쉽게 확장할 수 있다.