**Pintos Project 3: Threads**

담당 교수 : 김영재

조 / 조원 : 만점받으면행복하조 / 박민준

개발 기간 : 11/21 ~ 11/25

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

이번 프로젝트의 목표는 Pintos 운영체제에서 스레드와 스케줄링 관련 기능을 개선하고 확장하는 것이다.

1) Alarm Clock

- timer\_sleep() 함수를 개선하여 스레드가 지정된 시간 동안 효율적으로 대기하도록 구현한다. 대기 중인 스레드를 관리하기 위한 새로운 큐를 생성하여, timer interrupt가 발생했을 때 시간이 만료된 스레드만 ready 상태로 전환한다.

2) Priority Scheduling

- 스레드의 우선순위를 기반으로 CPU를 분배하는 스케줄러를 설계하여 시스템의 효율성을 향상시킨다. 높은 우선순위의 스레드가 등장하거나 ready 상태로 전환되었을 때 즉시 CPU를 선점하도록 구현한다. 낮은 우선순위 스레드가 실행되지 못하고 starvation에 빠지지 않도록 aging 기법을 도입하여 시간이 지남에 따라 우선순위가 증가하도록 한다.

3) Advanced scheduler

- BSD 알고리즘을 기반으로 Multi-Level Feedback Queue를 설계하고, 각 스레드의 recent\_cpu와 load\_avg 값을 계산하여 우선순위를 동적으로 조정한다. CPU를 많이 사용한 스레드의 우선순위를 낮추고, 시스템 부하를 효율적으로 분산시키는 구조를 구현한다Fixed-Point Arithmetic을 활용하여 부동소수점 연산 없이 필요한 계산을 수행하며, MLFQ와 함께 공정한 스케줄링을 보장한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock

- 기존 timer\_sleep() 함수는 대기 중인 스레드를 효율적으로 관리하지 못하며 스레드가 반복적으로 ready와 running 상태를 전환해 CPU 자원을 낭비한다. 이러한 비효율은 특히 여러 스레드가 동시에 실행되는 환경에서 성능 저하로 이어질 수 있다.

- 개선된 timer\_sleep() 함수는 대기 중인 스레드를 blocked 상태로 유지하고 깨어날 시간에만 ready 상태로 전환한다. 이를 통해 CPU 사용량이 최적화되며 시스템의 전반적인 성능이 향상된다. 또한, timer interrupt를 활용해 스레드 상태 전환이 더 정교하게 관리될 수 있다.

* 1. Priority Scheduling

- Pintos의 기본 Round-Robin 스케줄러는 스레드의 우선순위를 고려하지 않으며 모든 스레드에 동일한 자원을 배분한다. 이는 중요한 작업이 지연되거나 낮은 우선순위의 스레드가 높은 우선순위의 작업을 방해하는 상황을 초래할 수 있다.

- 우선순위 기반 스케줄링을 통해 CPU 자원을 효율적으로 배분할 수 있다. Aging 기법을 추가하면 낮은 우선순위의 스레드도 적절히 실행될 수 있어 starvation을 방지할 수 있다. 또한 ready\_list를 우선순위에 따라 정렬하여 스케줄링의 공정성과 효율성을 보장할 수 있다.

* 1. Advanced Scheduler (추가구현을 한 경우)

- 기본 우선순위 스케줄러는 정적 우선순위와 간단한 스케줄링 알고리즘만을 지원하며 시스템 부하가 증가하거나 복잡한 작업 부하를 처리하기에는 한계가 있다.

- BSD 스케줄러는 recent\_cpu와 lad\_avg를 활용한 동적 우선순위 계산을 통해 이러한 한계를 극복할 수 있다. BSD 스케줄러는 MLFQ 기반으로 스레드를 관리하여 다양한 우선순위와 작업 부하를 효과적으로 처리할 수 있다. 또한, CPU 사용량이 높은 스레드의 우선순위를 자동으로 낮추어 시스템 자원의 공정한 배분이 가능하다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

- 먼저, sleep\_list라는 리스트에 Blocked 상태의 스레드를 저장한다. 이후 현재 시간(ticks)과 깨어날 시간(wakeup\_tick)을 비교하여 시간이 만료된 스레드를 찾아 ready 상태로 전환한다. 이러한 함수를 thread\_awake() 함수로 선언한다.

- 시간이 만료된 스레드는 list\_remove()로 sleep\_list에서 제거되며 thread\_unblock() 함수를 호출해 ready\_list로 이동한다.

- 인터럽트 핸들러(timer\_interrupt())에 의해 주기적으로 thread\_awake() 함수가 호출되어 스레드 상태를 갱신한다.

- 즉, Blocked 상태에 있는 스레드의 wakeup\_tick이 현재 시간(ticks)보다 작거나 같다면 해당 스레드는 ready 상태로 전환된다. ready 상태로 전환된 스레드는 이후 스케줄러가 실행하도록 대기한다.

1. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.

- 먼저, ready\_list에서 최고 우선순위를 가진 스레드를 찾는다.

- 현재 실행 중인 스레드(thread\_current())의 우선순위와 비교하여 ready\_list의 스레드가 더 높은 우선순위를 가졌다면 thread\_yield() 함수를 호출하여 현재 스레드를 ready 상태로 전환한다.

- 이후 스케줄러가 실행되어 ready\_list의 가장 높은 우선순위 스레드를 실행한다.

- 즉, ready\_list의 스레드 중 가장 높은 우선순위를 가진 스레드가 실행 중인 스레드보다 높으면 즉시 선점한다. 이를 통해 CPU가 높은 우선순위 작업에 우선적으로 할당된다.

1. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)

- Advanced Scheduler에서 우선순위 계산식은 다음과 같다.

1. PRI\_MAX: 스레드가 가질 수 있는 최대 우선순위 (기본값=63)
2. recent\_cpu: 스레드가 최근에 사용한 CPU 시간으로, 많이 사용한 스레드의 우선순위를 낮추는 데 사용된다. recent\_cpu 계산식은 다음과 같다.

- load\_avg는 시스템 전체의 실행 대기 상태 스레드 수를 평균적으로 나타내며 스케줄링의 동적인 조정을 돕는 지표다. 시스템 상태는 매초 업데이트되며 계산식은 다음과 같다.

- 여기서 실행 대기 상태에 있는 스레드 수인 ready\_threads는 idle 스레드를 제외하고 계산된다.

1. nice: 스레드의 CPU 점유 성향을 나타내는 값으로 -20 ~ 20 사이의 정수이다. 높은 nice 값은 우선순위를 낮추어 다른 스레드에게 CPU 사용 기회를 준다.

- 최근 CPU 사용량(recent\_cpu)과 nice 값을 기반으로 우선순위를 동적으로 계산한다. 부동소수점 연산을 지원하지 않으므로 고정소수점 연산을 사용하여 recent\_cpu와 load\_avg를 계산한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11/21: proj3 명세서 읽고 내용 이해

11/22: Alarm Clock, Priority Scheduling 구현

11/23: Advanced Scheduler 구현

11/24~25: 보고서 작성

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야 하는 소스코드

**1. Alarm Clock**

[src/devices/timer.c]

- timer\_sleep():현재 스레드를 지정된 시간 동안 Blocked 상태로 만들고 지정된 시간이 지나면 다시 ready 상태로 복귀시키는 로직을 구현해야 한다. 이 과정에서 sleep\_list와 wakeup\_tick을 활용하였고, 스레드가 깨어날 시점을 계산하여 thread\_current()의 wakeup\_tick 필드에 저장한다. 이후 현재 스레드를 sleep\_list에 삽입하고, 현재 스레드가 실행되지 않도록 Blocked 상태로 전환하기 위해 thread\_block() 함수를 호출한다.

- timer\_interrupt(): sleep\_list에 있는 모든 스레드를 순회하며 wakeup\_tick 값이 현재 tick 값보다 작거나 같은 스레드를 찾는다. 이러한 스레드는 대기 시간이 완료된 상태이다. sleep\_list에서 제거된 스레드는 ready 상태로 복귀해야 하므로 thread\_unblock() 함수를 호출하여 ready\_list에 추가한다.

**2. Priority Scheduling**

[src/threads/thread.c]

- thread\_create(): 새로 생성된 스레드의 우선순위가 현재 실행 중인 스레드의 우선순위보다 높다면 CPU를 선점하도록 수정한다. ready\_list에 삽입된 스레드와 현재 실행 중인 스레드의 우선순위를 비교한 후, 필요한 경우 thread\_yield() 함수를 호출한다.

- thread\_unblock(): ready\_list에 스레드를 추가할 때 우선순위에 따라 정렬된 위치에 삽입하도록 수정한다. 기존의 list\_push\_back() 함수 대신 list\_insert\_ordered() 함수를 사용하여 ready\_list를 항상 우선순위 기준으로 유지한다.

- thread\_yield(): ready\_list에 현재 스레드를 추가할 때 우선순위에 따라 정렬된 위치에 삽입한다. thread\_unblock() 함수와 마찬가지로 list\_insert\_ordered() 함수를 사용하여 우선순위를 고려한 리스트 관리를 수행한다.

[src/thread/synch.c]

- sema\_up(): 스레드가 ready 상태로 전환된 후 현재 실행 중인 스레드보다 우선순위가 높다면 선점하도록 test\_max\_priority() 함수를 호출한다.

**3. Advanced Scheduler**

[src/threads/thread.c]

- thread\_init(): 모든 스레드가 nice 값과 recent\_cpu 값을 가지도록 초기화한다.

- thread\_start(): 스케줄러가 시작되기 전에 load\_avg를 기본값으로 초기화한다.

- thread\_tick(): thread\_prior\_aging 값이 true라면 thread\_aging() 함수를 호출하고, thread\_mlfsqs 값이 true라면 thread\_mlfqs\_tick() 함수를 호출한다.

- thread\_set\_priority(): 스레드의 우선순위를 새 값으로 설정하고 우선순위가 낮아진 경우 즉시 스케줄링을 수행한다.

- thread\_set\_nice(): 스레드의 nice 값을 설정하고 우선순위를 갱신한다. ready\_list가 비어 있지 않다면 해당 list의 최고 우선순위 스레드와 비교하여 스케줄링 여부를 결정한다.

- thread\_get\_nice(): 현재 스레드의 nice값을 반환한다.

- thread\_get\_recent\_cpu(): 현재 스레드의 recent\_cpu 값에 100만큼 곱한 후 정수 형태로 반환한다.

- thread\_get\_load\_avg(): 현재 스레드의 load\_avg 값에 100만큼 곱한 후 정수 형태로 반환한다.

* + 수정하거나 추가해야 하는 자료구조

[src/threads/thread.c]

**-** int load\_avg: 시스템 부하를 나타내는 값으로 시스템의 ready 상태 스레드 수를 기반으로 매초 갱신된다.

**-** bool thread\_mlfqs: MLFQS 활성화 여부를 결정하는 플래그이다. 이 값이 true로 설정되면 Advanced Scheduler를 사용하며 스레드의 우선순위는 정적으로 설정되지 않고 recent\_cpu와 nice 값을 기반으로 동적으로 계산된다.

**-** bool thread\_prior\_aging: Priority Aging 활성화 여부를 결정하는 플래그이다. 이 값이 true인 경우 오래 대기 중인 스레드의 우선순위가 증가하여 CPU 할당 기회를 높인다.

[src/threads/thread.h]

- struct 구조체: int64\_t 형태의 wakeup\_tick, recent\_cpu, nice 값을 추가하였다.

* + 수정하거나 추가해야 하는 함수

[src/threads/thread.c]

- cmp\_priority(): 두 스레드의 우선순위를 비교하는 함수다. 해당 함수는 우선순위 스케줄링 및 ready\_list 정렬에 사용된다. 우선순위 값이 더 높은 스레드가 앞으로 오도록 정렬한다.

- test\_max\_priority(): 현재 실행 중인 스레드의 우선순위를 ready\_list의 최고 우선순위 스레드와 비교한다. ready\_list의 최고 우선순위 스레드가 현재 스레드보다 우선순위가 높으면 CPU를 양보한다.

- mlfqs\_load\_avg(): load\_avg값을 계산한다. ready 상태 스레드 수를 기반으로 매초마다 업데이트된다.

- mlfqs\_increment(): 현재 스레드의 recent\_cpu 값을 1 증가시킨다.

- mlfqs\_recent\_cpu(): 스레드의 recent\_cpu 값을 계산한다. load\_avg와 nice 값을 기반으로 갱신한다.

- mlfqs\_recalc\_recent\_cpu(): 모든 스레드의 recent\_cpu 값을 업데이트한다.

- mlfqs\_priority(): 스레드의 우선순위를 계산한다. recent\_cpu와 nice값을 기반으로 갱신한다.

- mlfqs\_recalc\_priority(): 모든 스레드의 우선순위를 갱신한다. ready\_list의 최고 우선 순위스레드와 현재 스레드를 비교하여 필요 시 선점한다.

- thread\_mlfqs\_tick(): MLFQS 관련 작업을 매 tick마다 처리한다. 현재 스레드의 recent\_cpu 값을 증가시키고, 매초 load\_avg와 모든 스레드의 recent\_cpu를 갱신한다.

- thread\_aging(): 대기 중인 스레드의 우선순위를 증가시켜 오래 대기한 스레드가 CPU를 얻을 가능성을 높인다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)

**1. Alarm Clock**

**텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2. Priority Scheduling**

**텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명**

* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)

**1. Alarm Clock**

[src/devices/timer.c]

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**-** timer\_ticks() 함수를 호출하여 현재 시각을 얻는다. start + ticks를 계산해 현재 시각부터 지정된 tick 수만큼 대기 후 깨워야 하는 시점을 설정한다.

**-** 인터럽트가 활성화된 상태에서는 스레드 상태를 안전하게 변경할 수 없으므로 intr\_disable()을 호출해 인터럽트를 비활성화한다. 이후 현재 스레드를 sleep\_list에 추가한다. 이 리스트는 wakeup\_tick을 기준으로 관리된다.

**-** thread\_block() 함수를 호출하여 스레드 상태를 Blocked로 변경한다. 스케줄러는 다른 ready 상태의 스레드를 실행시킨다. 스레드 상태 변경이 완료되면 intr\_set\_level()로 인터럽트를 원래 상태로 복원한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- sleep\_list를 순회하여 깨워야 할 tick 값이 도달한 스레드를 찾는다. wakeup\_tick 값이 현재 tick 수와 같거나 작으면 해당 스레드를 list\_remove()로 대기 리스트에서 제거한다.

- thread\_unblock() 함수를 호출하여 Blocked 상태였던 스레드를 ready 상태로 전환한다. 해당 스레드는 ready\_list에 추가되며 스케주러에 의해 실행될 준비가 된다.

- 마지막에 thread\_tick() 함수를 호출하여 스레드와 관련된 tick 기반 작업을 수행한다.

**2. Priority Scheduling**

[src/threads/thread.c]

**텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명**

**[thread\_create() 함수 밑부분]**

- 우선순위 기반 선점 스케줄링 지원을 위해 스레드를 생성한 후 현재 스레드와 ready\_list의 최고 우선순위 스레드를 비교한다. 만약 현재 스레드의 우선순위가 더 낮다면 thread\_yield() 함수를 호출하여 CPU를 양보한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- Blocked 상태의 스레드를 ready 상태로 전환하고 ready\_list에 추가한다. ready\_list에 추가 시 우선순위 기반으로 정렬되도록 list\_insert\_ordered 함수와 cmp\_priority 함수를 사용한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 현재 실행 중인 스레드를 ready 상태로 전환하고 스케줄링을 요청한다. ready\_list에 삽입할 때 우선순위 기반으로 스레드를 삽입한다.

[src/thread/synch.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- ready\_list에서 우선순위가 가장 높은 스레드를 먼저 깨우기 위해 리스트를 정렬한다. test\_max\_priority() 함수를 호출해 우선순위 스케줄링을 점검한다.

**3. Advanced Scheduler**

[src/threads/thread.h]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**-** thread 구조체에 int64\_t 형태의 wakeup\_tick, recent\_cpu, nice 변수를 추가한다.

[src/threads/thread.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- recent\_cpu와 nice 값을 초기화하여 MLFQS를 지원한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- thread\_start 함수에 초기 load\_avg 값을 설정한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- thread\_prior\_aging 활성화 시 스레드의 aging 작업을 수행한다.

- thread\_mlfqs 활성화 시 MLFQS 관련 작업을 수행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 현재 스레드의 우선순위를 설정한다. 만약 설정된 우선순위 값이 기존보다 더 작아졌다면 다시 스케줄링 작업을 진행한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 현재 실행 중인 스레드의 nice 값을 설정한다.

- MLFQS가 활성화된 상태에서 mlfqs\_priority() 함수를 호출하여 새로 설정된 nice 값을 기반으로 우선순위를 재계산한다. 이 단계에서 스레드의 recent\_cpu와 nice 값을 사용하여 동적 우선순위가 업데이트된다.

- test\_max\_priority() 함수를 호출하여 우선순위 값을 비교한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 현재 스레드의 nice 값을 반환한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- 시스템의 load\_avg 값에 100을 곱한 값을 정수 형태로 반올림하여 반환한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 현재 스레드의 recent\_cpu 값에 100을 곱한 값을 정수 형태로 반올림하여 반환한다.

* + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명

[src/threads/thread.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 현재 실행 중인 스레드의 우선순위와 ready\_list의 최고 우선순위 스레드를 비교하는 함수다.

- ready\_list가 비어있는 경우 더 이상 우선순위 비교나 스케줄링 작업을 진행할 필요가 없으므로 함수를 종료한다.

- ready\_list에서 가장 높은 우선순위를 가진 스레드를 확인한다. list\_front()를 통해 ready\_list의 첫 번째 요소를 가져오고 이를 스레드 구조체로 변환한다.

- 현재 실행 중인 스레드의 우선순위가 ready\_list의 최고 우선순위 스레드보다 낮은 경우 thread\_yield() 함수를 호출하여 현재 스레드가 CPU를 양보한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 두 스레드의 우선순위를 비교하여 정렬 기준을 제공하는 함수다. 이는 ready\_list 등에서 우선순위 기반 정렬을 구현할 때 사용된다.

- 리스트의 요소 포인터를 list\_entry()를 통해 struct thread로 변환한다.

- 비교 대상이 되는 두 스레드 thread\_a, thread\_b를 가져온다.

- thread\_a의 우선순위가 thread\_b보다 높으면 true를 반환한다. 이는 리스트 정렬 시 높은 우선순위의 스레드가 리스트의 앞쪽에 오도록 한다.

[src/threads/fixed\_point.h]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 해당 파일은 고정소수점 연산을 지원하기 위해 새로 작성한 파일이다. 이는 스레드의 우선순위와 CPU 사용량 계산 시 사용된다.

* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

- 우선순위 기반 스케줄링에서 현재 실행 중인 스레드가 ready\_list의 높은 우선순위 스레드보다 먼저 실행되는 경우가 발생하였다. 이는 thread\_create() 또는 sema\_up() 함수 호출 이후 ready\_list 정렬이 적절히 적용되지 않았기 때문이다. test\_max\_priority() 함수를 활용하여 해당 문제를 해결하였다.

- MLFQS 활성화 시 계산이 부정확하였다. recent\_cpu와 priority 계산에서 정수와 고정소수점 연산이 혼재되어 값이 정확히 계산되지 않았다. 이러한 문제는 계산 과정 중 타입 변환을 명시적으로 수행하는 int\_to\_fp, fp\_to\_int 함수를 사용함으로써 해결하였다.

- aging 처리 과정에서 ready\_list의 스레드만 대상으로 우선순위를 증가시켰다. Blocked 상태나 sleep 상태의 스레드는 고려하지 않았기 때문에 starvation 문제를 해결하지 못했다. 따라서 ready\_list에서 모든 스레드 리스트(all\_list)로 대상을 확장하여 모든 스레드의 우선순위를 증가시켰다.

* 1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- priority-lifo.c 파일은 우선순위에 따라 스레드가 실행되는 순서를 확인하는 테스트이다. 스레드들이 생성된 후, 각 스레드는 지정된 우선순위에 따라 실행되고, 결과가 동일한 순서로 반복되는지 검증한다. LIFO 방식으로 우선순위가 높은 스레드가 먼저 실행된다.

- 테스트 결과 총 16개의 스레드가 생성되고 각 스레드는 16번씩 반복 실행되었다. 출력 포맷은 ‘(priority-lifo) interation:’으로 시작하여 스레드의 id가 출력되었고, 높은 우선순위(15) 스레드부터 낮은 우선순위(0) 스레드까지 실행되었다.

* make check 수행 결과를 캡처하여 첨부

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- cspro에서 make check 수행 시 수행 시간이 매우 오래 걸려 정확한 수행 결과를 확인할 수 없었다. 따라서 vscode를 이용해 local 환경에서 작업한 후 make check 수행 결과를 확인하였고, 이를 캡쳐하여 첨부하였다.