Operating Systems

Spring, 2022

School of Software, CAU

**Project #1**

**- Thread Systems and Synchronization –**

Due: Midnight, May 3th, 2022

**1. 프로젝트 개요**

Pintos는 OS를 수강하는 학부생들이 OS의 주요 구성 요소인 커널 스레드, 메인 메모리 관리, 파일 시스템 등을 이해할 수 있도록 설계 및 구현된 교육용 OS이다. Pintos는 80x86 아키텍처 상에서 동작할 수 있도록 설계되었지만, Project #1에서는 개발의 편의성을 위해 오픈 소스 CPU 에뮬레이터인 QEMU(Quick EMUlator)를 기반으로 한 가상 머신(Virtual Machine(VM)) 상에서 실행되는 것으로 가정한다.

Project #1의 목적은 여러분이 Pintos가 제공하는 스레드 시스템(강의록 4장 참조)과 세마포어 등 동기화 장치들(강의록 5장 참조)을 이해하고, 동기화 관련 문제를 해결하는 데 있다. 이를 위해 우선 Pintos ‘threads’ 폴더 내의 스레드 생성, 라운드 로빈 스케줄링, 세마포어, 뮤텍스, 모니터 등 스레드 시스템 및 동기화 장치들에 관련된 소스 코드(thread.h, thread.c, synch.h, synch.c 등)를 이해해야 한다.

Project#1 수행을 위해 필요한 파일들은 ‘threads’ 및 ‘projects’ 폴더 내에 있으며, 이들의 기능은 [표 1]에 요약되어 있다.

[표 1] ‘threads’ 폴더 내 파일과 기능

|  |  |
| --- | --- |
| loader.S, loader.h | PC BIOS가 디스크로부터 메모리에 적재하는 512바이트 크기의 커널 로더로서 로딩 후 ‘start.S’ 내의 함수 ‘start()’를 호출 |
| start.S | 커널의 일부로서 80x86의 메모리 보호와 32비트 작업 실행을 위한 초기 동작 수행 |
| kernel.lds.S | 커널 링크 스크립트 |
| init.c, init.h | 커널 진입점으로서 함수 ‘main()’을 포함한 초기화 작업을 수행하며, Project #1 실행을 위해 필요한 코드 호출부를 포함 |
| thread.c, thread.h | 스레드 지원 코드로서 스레드 생성, 스케줄링 등 스레드 시스템 지원을 위한 기본 기능 지원 |
| synch.c, synch.h | 동기화 기능 지원 코드로서 세마포어, 뮤텍스 및 모니터 등 동기화를 위한 기능 지원 |
| switch.S, switch.h | 스레드 간 컨텍스트 스위칭 (Context Switching) 지원을 위한 어셈블리 코드 |
| palloc.c, palloc.h | 4KB 크기의 페이지 할당을 위한 시스템 메모리 관리자 |
| malloc.c, malloc.h | 커널 함수로서 메모리 할당 및 반환을 위한 함수 ‘malloc()’ 및 ‘free()’ 포함 |
| interrupt.c, interrupt.h | 인터럽트 ON/OFF를 위한 기본 인터럽트 핸들러 및 관련 함수 |
| intr-stubs.S,  intr-stubs.h | 저수준 (Low-level) 인터럽트 핸들러를 위한 어셈블리 코드 및 관련 헤더 파일 |
| io.h | ‘devices’ 폴더 내의 코드를 이용해 I/O 포트 접근을 지원하는 함수 호출에 필요한 헤더 파일 |
| vaddr.h, pte.h | 가상 주소 및 페이지 테이블 엔트리(PTE) 관련 작업을 지원하는 함수 및 매크로 호출에 필요한 헤더 파일 |
| flags.h | 80x86 ‘flags’ 레지스터의 각 비트를 정의하는 매크로 호출에 필요한 헤더 파일 |

**2. 프로젝트 수행 준비**

**(1) Pintos 설치**

Project #1에서는 GCC 3.3 이상의 컴파일러를 이용해 80x86 상에서 Pintos를 컴파일한 뒤, 여러분이 작성한 코드를 QEMU 기반 VM 상에서 실행하는 것으로 가정한다. 여러분은 작성한 코드를 Hypervisor 기반 VM (예: VMware Workstation Player, VirtualBox 등)에 리눅스를 설치해 실행하거나, Windows Subsystem for Linux (WSL), 또는 Docker를 이용한 리눅스 컨테이너에서 실행할 수 있다. 본 프로젝트에서 사용된 코드의 개발 및 테스트는 GCC7.5.0(64bit Ubuntu 18.04.3)에서 수행되었다.

※ Ubuntu Linux 18.04 LTS [[링크](https://mirror.kakao.com/ubuntu-releases/bionic/)]

※ VMware Workstation Player [[링크](https://www.vmware.com/go/getplayer-win)]

**(2) Pintos 빌드 및 실행**

Pintos 빌드를 위해서는 Ubuntu Linux 환경에서 아래 명령어를 사용하여 GCC, GNU Binutils, Perl, GNU make, 그리고 QEMU 패키지를 설치한다.

|  |
| --- |
| # sudo apt install build-essential perl qemu |

※ Project #1을 위한 프로젝트 아카이브(‘cau15841-pintos-qemu\_v2.tgz’)

패키지 설치가 완료되면 아래에서 와 같이 프로젝트 아카이브(‘cau15841-pintos-qemuv2.tgz’)를 다운로드 받아 원하는 곳에 압축을 풀고(㉠), 해당 프로젝트 폴더(pintos\_qemu)로 이동(㉡, ㉢)한 다음, Pintos의 빌드(㉣)와 실행(㉤)이 가능한지 확인한다.

|  |
| --- |
| # 프로젝트 폴더로 이동  # tar -zxvf cau15841-pintos-qemuv2.tgz --------------------- ㉠  # cd pintos\_qemu -------------------------------- ㉡  # cd threads --------------------------------------------- ㉢  # make --------------------------------------------------- ㉣  # cd build  # ../../utils/pintos crossroads aAC:bBD:cCD:dDB ---------- ㉤ |

위 명령어 “../../utils/pintos crossroads aAC:bBD:cCA:dDB”에서 첫 번째 파라미터는 교차로 문제 실행을 위해 필요한데, ‘threads/init.c’에서 ‘projects/crossroads/crossroads.h’에 구현된 교차로 문제를 호출한다.

**(3) 소스코드 추가 방법**

Project #1 수행을 위해 필요한 코드는 ‘pintos\_qemu/projects/crossroads’ 폴더 내에 추가되어야 한다. 예를 들어, ‘mycode.c’라는 소스 코드를 추가하고자 한다면, ‘pintos\_qemu/projects/crossroads’ 폴더 내의 ‘Make.projects’ 파일의 10번 라인과 같이 추가를 하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

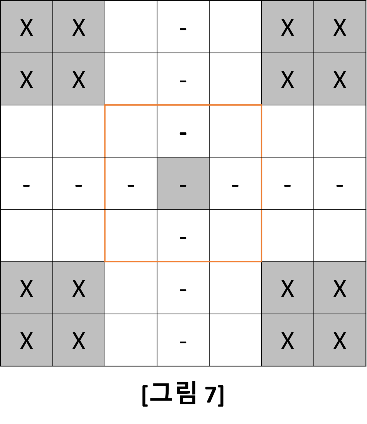
**3. 프로젝트 설명**

**(1) 교차로 문제**

교차로 문제의 목적은 [그림 1~6]과 같은 **사거리** **교차로에 접근하는 차량들**이 충돌하지 않고, 교착상태 (Deadlock) 발생 없이 주행할 수 있도록 제어하는 코드를 구현하는데 있다. 이 문제에서 **모든 차량은 개별적인 스레드로 동작**해야 하며, **각 단위 스텝에 최대 한 칸을 전진**해야 한다. 다시 말해, 모든 전진 가능한 차량 스레드가 한 칸 전진하고 나서 단위 스텝이 1씩 증가해야 한다. 제공되는 소스코드에서는 단위 스텝이 구현되어 있지 않고, 단순히 sleep 함수를 이용해 1초마다 한 칸씩 전진하도록 구현되어 있다. 단위 스텝에 전진 가능한 차량이 아직 모두 전진하지 않은 경우 단위 스텝은 증가하지 않아야 한다. ‘projects/crossroads/’ 폴더 내에서 제공되는 소스코드를 통해 차량들이 어떻게 개별적으로 동작하도록 구현되어 있는지 이해하기 바란다. 단위 스텝 동작을 구현하여 ‘crossroads.h’ 헤더 파일에 정의된 ‘***int crossroads\_step***’ 변수에 저장하도록 하라. 차량들이 단위 스텝 당 한 칸씩 전진할 수 있도록 제어하기 위해 강의록 5장에서 설명한 동기화 문제 해결 방법을 참고하기 바란다.

도로 A를 이용하여 하나의 교차로에 도착한 차량은 세 갈래 길 중 하나로 진행할 수 있다([그림 1] 참조). 이 교차로에서는 **차량이 교차로에 진입하면 경로가 겹치는 다른 차량의 진입이 제한된다**. 예를 들어, 도로 A에 있는 차량이 도로 C로 진행한다면 도로 C에 있는 차량은 도로 A, D로는 진행이 가능하지만([그림 2] 참조]), 도로 B로는 진행할 수 없다([그림 3] 참조]). 마찬가지로 [그림 4]에서처럼 도로 B에 있는 차량이 도로 A로 진행한다면, 도로 A에 있는 차량이 도로 B로는 진행이 가능하지만, 도로 C, D로는 진행할 수 없다. 또 다른 예로서, [그림 5]와 같이 도로 A에 있는 차량이 도로 D로 진행하는 경우 B의 차량과 C의 차량은 진입이 불가능하고, 도로 D의 차량은 도로 A로 진행 가능하다. 또한 [그림6]과 같이 도로 A에 있는 차량이 도로 B로 진행을 할 때, 도로 B에 있는 자동차는 도로 C나 D로, 도로 C에 있는 자동차는 도로 A나 D로 진행이 가능하다.

|  |  |  |
| --- | --- | --- |
| [그림 1] | [그림 2] | [그림 3] |
| [그림 4] | **[그림 5]** | [그림 6] |



교차로 문제에서 경로를 정의하기 위해 [그림 7]과 같은 7 x 7 그리드를 사용한다. 전체 도로는 세마포어를 통해 한 칸에 한 대의 차량이 점유할 수 있도록 구현되어 있다.

**입력 데이터**는 연속된 3개의 알파벳을 하나의 데이터 단위로 하여 각 알파벳 단위가 ‘:’ 기호로 구분된 문자열이다. 각 알파벳 단위는 **VehicleID, Start, Dest로 구성**되다.여기서 **VehicleID**는 알파벳 소문자 a-z, **Start** 및 **Dest**는 각각 알파벳 대문자 A-D로 표시된다. 예를 들어, “aAC:bBA” 문자열을 입력하면 차량 ‘a’는 도로 A에서 도로 C로 진행할 예정이며, 차량 ‘b’는 도로B에서 도로 A로 진행할 예정임을 의미한다. 각 차량 스레드는 임의의 시점에 생성될 수 있으나, 하나의 차량은 **각 단위 스텝에 한 칸 씩만 전진**할 수 있도록 해야 한다.

**(2) 구현 시 요구사항 요약**

1. ‘projects/crossroads’ 폴더 내에 있는 소스코드를 수정하거나 새로운 소스코드를 추가하여 교차로 문제를 해결할 것.
2. 각 차량 스레드는 **단위 스텝 당 최대 한 칸 전진**할 수 있도록 제한하되, 모든 차량이 최대 **한 칸 전진할 때까지 단위 스텝이 증가하지 않아야 함**. 단위 스텝은 반드시 ‘crossroads.h’ 헤더 파일에 정의된 ‘***int crossroads\_step***’ 변수에 저장되어야 함.
3. 반드시 **이동할 수 있는 모든 차량**이 이동한 후에 단위스텝을 증가시켜야 함.
4. ‘***int crossroads\_step***’ 변수는 항상 **증가하여야 하며, 감소할 수 없음**.
5. ‘***int crossroads\_step***’ 변수가 **변경될 때 마다** ‘ats.h’에 정의된 ***unitstep\_changed()*** 함수를 호출하여야 하며, 이 함수는 **모든 스레드로부터 보호**되어야 한다. (***unitstep\_changed()***는 공유 리소스에 대한 접근보호가 보장되어야 함.)
6. 각 차량 스레드가 교차로에 진입하는 방식을 제어하여 **잠재적으로 발생 가능한 교착상태 문제를 해결**해야 함.
7. 작성한 코드에서 **임의의 시간대기가 발생하지 않도록** 해야함. (***timer\_msleep()*** 과 같은 방식을 사용하지 말 것.)

**(3) 프로젝트 제출물 및 주의사항**

|  |
| --- |
| **[프로젝트 제출물]**  1. 프로젝트 폴더 압축 전, ‘make clean’ 명령어를 이용하여 빌드 파일들을 모두 제거한 결과  2. 교차로 문제에서 사용한 동기화 방법 및 결과를 설명하는 프로젝트 보고서  3. 프로젝트 결과물(소스코드 + 프로젝트 보고서)은 압축하여 e-클래스를 통해 제출 |
| **[주의사항]**   1. 새로운 파일 추가는 ‘projects/crossroads’폴더 내로만 제한됨. 2. ‘projects/crossroads’ 폴더 내 ‘crossroads.c’ 및 ‘crossroads.h’ **소스코드는 수정할 수 없음**. 3. ‘projects/crossroads’ 폴더 내 ‘ats.c’ 및 ‘ats.h’ **소스코드는 수정할 수 없음**. 4. 단위 스텝인 ***crossroads\_step*** 변수가 수정될 경우 반드시 ats.h에 정의되어있는 ***unitstep\_changed()*** 함수를 호출할것. 단 이 함수는 **스레드로부터 보호**되어야 하며 ***crossroads\_step*** 변수가 **수정될 때 한번 만 호출**되어야 함. |