Operating Systems

Spring, 2018

School of Software, CAU

**Project #1**

**- Thread Systems –**

Due: 2018년 4월 18일 오전 0시

**1. 프로젝트 개요**

80x86 아키텍처를 위한 교육용 운영체제인 Pintos는 커널 스레드, 유저 프로그램 적재 및 실행, 파일 시스템 등을 실제 운영체제에 비해 이해하기 쉽게 매우 간단히 구현한 것이 특징입니다. Pintos는 80x86 아키텍쳐 상에서 동작할 수 있도록 설계되었지만, 프로젝트 #1에서는 개발의 편의성을 위해 오픈 소스 하드웨어 에뮬레이터인 QEMU(Quick Emulator)를 통한 가상 머신 환경에서 실행하는 것을 가정합니다.

Project #1에서 여러분은 Pintos에서 일부분만 제공될 스레드 시스템을 완성한 다음, 이 시스템을 이용하여 아래에서 요구하는 두개의 동기화 관련 문제를 해결해야 합니다. 이를 위해 우선 ‘threads’ 디렉토리 내에 있는 일부분만 완성된 스레드 시스템을 이해하는 것이 필요합니다. 제공되는 스레드 시스템에는 Thread Fork, 단순 라운드 로빈 스케줄링, 동기화를 위한 세마포어 등이 이미 구현되어 있습니다.

만일 여러분이 작성한 코드가 제대로 동기화 되었다면, Pintos의 스케줄러가 Ready Queue에 있는 스레드를 어떤 순서로 실행시켜도 정상적으로 동작할 것입니다. 다시 말해, OS 스케줄러의 Ready Queue에 스레드를 추가하기 위해 인터럽트 발생이 허용된 여러분 코드 임의의 위치에 우리가 ‘thread\_yield()’ 함수를 삽입하더라도 여러분의 코드가 항상 정확하게 동작하도록 해야 합니다.

Pintos 에서는 각 스레드에 4KB 미만의 작은 고정 크기 스택이 할당됩니다. 커널이 스택 오버플로우를 감지하는 기능을 가지고 있지만 완벽하게 수행되지 않기 때문에 ‘int buf[1000]’ 등의 Large Non-static 지역 변수를 선언하는 경우 알 수 없는 커널 패닉 현상 등이 발생할 수 있습니다.

이번 프로젝트를 위해 여러분들이 사용할 파일들은 ‘threads’ 디렉토리 내에 있으며 이들의 역할은 [표 1]과 같습니다. 프로젝트를 위해 참고해야 할 코드는 **밑줄로** 표시하였습니다. 더 자세한 설명은 링크[[Pintos Project](https://web.stanford.edu/class/cs140/projects/pintos/pintos.pdf)]를 참고해 주시고, 추가적으로 시간 동기화를 위해 ‘devices’ 디렉토리 내의 ‘timer.c’, ‘timer.h’ 함수를 파악할 필요가 있습니다.

[표 1] ‘thread’ 디렉토리

|  |  |
| --- | --- |
| loader.S loader.h | PC BIOS가 디스크로부터 메모리에 적재하는 512바이트 크기의 커널 로더로서 로딩 후 ‘start’S’ 내의 start() 함수를 호출 |
| start.S | 커널의 일부로서 80x86의 메모리 보호와 32비트 작업을 위한 설정을 진행 |
| kernel.lds.S | 커널 링크 스크립트 |
| **init.c init.h** | **커널 진입점으로서 main()을 포함한 초기화 작업을 수행하며, 프로젝트 #1 코드 호출부를 포함** |
| **thread.c thread.h** | **스레드 지원 코드로서 Thread Fork, Scheduling 등의 기본 기능을 지원** |
| **synch.c synch.h** | **동기화 기능 지원 코드로서 프로젝트 #1에서는 동기화를 위한 세마포어가 미리 구현되어 있음** |
| switch.S switch.h | 스레드 컨텍스트 스위칭 지원을 위한 어셈블리 코드 |
| palloc.c palloc.h | 4KB 페이지 할당을 위한 시스템 메모리 관리자 |
| malloc.c malloc.h | 커널을 위한 malloc(), free() 함수 |
| interrupt.c interrupt.h | 인터럽트 ON/OFF를 위한 기본 인터럽트 핸들러 및 관련 함수 |
| intr-stubs.S  intr-stubs.h | 저수준 인터럽트 핸들러를 위한 어셈블리 코드 및 관련 헤더 파일 |
| io.h | ‘devices’ 디렉토리 내의 코드를 통해 I/O 포트 접근을 지원하는 함수 호출에 필요한 헤더 파일 |
| vaddr.h pte.h | 가상 주소 및 페이지 테이블 엔트리(PTE) 작업을 지원하는 함수 및 매크로 호출에 필요한 헤더 파일 |
| flags.h | 80x86 ‘flags’ 레지스터 비트를 정의하는 매크로 호출에 필요한 헤더 파일 |

여러분은 프로젝트 #1에서 **synch.c** 및 **synch.h 파일을 수정해야 하며, projects 디렉토리 내에 필요한 소스 코드를 추가**로 개발해야 합니다.

**2. 프로젝트 수행 준비**

**(1) Pintos 설치 준비**

프로젝트 #1는 80x86 호스트에서 GCC 3.3 이상의 컴파일러를 통해 Pintos를 컴파일한 뒤, QEMU 환경에서 여러분이 작성한 코드를 실행하는 것을 가정합니다. WSL(Windows Subsystem for Linux)이나 리눅스 배포판을 이용한 리눅스 개발 환경을 구축하여 사용하는 것을 추천합니다. 원하는 경우, 약간의 추가 설정 후, macOS 에서도 컴파일 및 실행이 가능합니다. 실제 개발 및 테스트는 GCC4.7(WSL 환경) 및 GCC7.2(Ubuntu 17.10 amd64 환경)에서 이루어질 것입니다.

Project #1 수행을 위해서 Hypervisor 기반 가상 머신 소프트웨어인 VMware Workstation Player를 이용해 리눅스 가상 머신이 설치된 개발 환경을 구축해야 합니다. 이 소프트웨어는 윈도우 상에서 리눅스 운영체제를 실행할 수 있도록 지원합니다. 리눅스 배포판 및 VMware Workstation Player 설치를 위해서는 아래 자료를 다운로드 받아 사용하기 바랍니다.

※ Ubuntu Linux 17.10 amd64 [[다운로드](http://ftp.daumkakao.com/ubuntu-releases/17.10.1/ubuntu-17.10.1-desktop-amd64.iso)]

※ VMware Workstation Player [[다운로드](https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/14_0)]

※ 설치 방법 다운로드 [[다운로드](https://drive.google.com/file/d/1iGqCYiss6MQTBUrdhojIR9qKgY76nYQ5/view?usp=sharing)]

**(2) 빌드 환경 설정**

Pintos 빌드를 위해서는 먼저 GCC, GNU binutils, Perl, GNU make, QEMU 패키지가 설치되어 있어야 합니다. 이를 위해 Ubuntu Linux 환경에서 아래 명령어를 사용하면 됩니다.

|  |
| --- |
| # sudo apt install build-essential gcc binutils perl qemu gdb |

**(3) Pintos 빌드 및 실행**

※ cau15841-pintos-qemu 아카이브 [[다운로드](https://drive.google.com/file/d/12rlxkQk_tgw-qIbqgJ2xRQXm3Q5UJdNc/view?usp=sharing)]

패키지 설치가 완료되면 프로젝트 아카이브를 다운로드 받습니다. 적당한 곳에 압축을 푼 다음, 해당 프로젝트 디렉토리로 이동한 뒤, 빌드 및 실행을 위해 아래와 같은 명령어를 입력합니다.

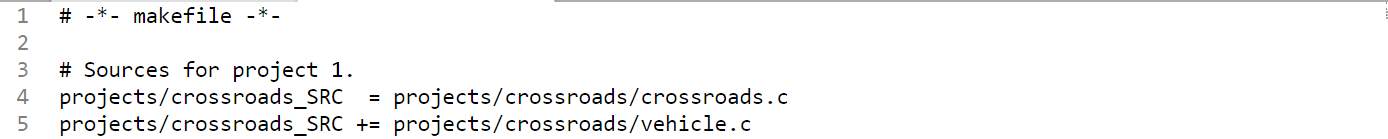
|  |
| --- |
| # 프로젝트 디렉토리로 이동합니다  # cd threads  # make  # cd build  # ../../utils/pintos crossroads aAB:bBC:cCA |

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

**(4) 소스코드 추가 방법**

여러분은 프로젝트 #1 수행을 위한 코드를 추가해야 합니다. 이를 위해 Pintos 디렉토리 내에 ‘projects’ 라는 디렉토리가 이미 생성되어 있습니다. 또한 여러분의 편리를 위해 ‘projects’ 디렉토리 내에 project #1에서 구현 해야할 Message Passing 코드 테스트를 위한 디렉토리(msgpassing)와 교차로 문제 해결을 위한 코드 테스트를 위한 디렉토리(crossroads)를 미리 생성했습니다.

만약 여러분이 ‘projects/crossroads’ 디렉토리 내에 ‘vehicle.c’라는 이름을 가진 소스코드를 추가하고자 한다면 ‘projects/crossroads’ 디렉토리 내의 Make.projects 파일을 아래와 같이 편집해야 합니다.



**3. 프로젝트 수행**

**(1) Message Passing** Project #1에서 여러분은 미리 제공된 Semaphores를 사용하여 **‘threads’ 디렉토리 내 ‘synch.c’와 ‘synch.h’에 Blocking Send 및 Blocking Receive를 지원하는 Message Passing 기능을 추가해야** 합니다. 또한 Message Passing의 정상 동작 확인을 위한 테스트 코드를 ‘projects/msgpassing/’ 디렉토리에 구현하고, 테스트 방법과 결과를 보고서에 기술하십시오. 테스트 코드는 ‘threads/build/’ 폴더에서 아래 명령어를 이용하여 실행이 가능합니다.

|  |
| --- |
| # ../../utils/pintos messagepassing |

Pintos 시작 명령어로 빈 테스트 함수를 실행하도록 ‘init.c’가 수정되어 있으니 필요시 활용하기 바랍니다.

**(2) 교차로 문제**

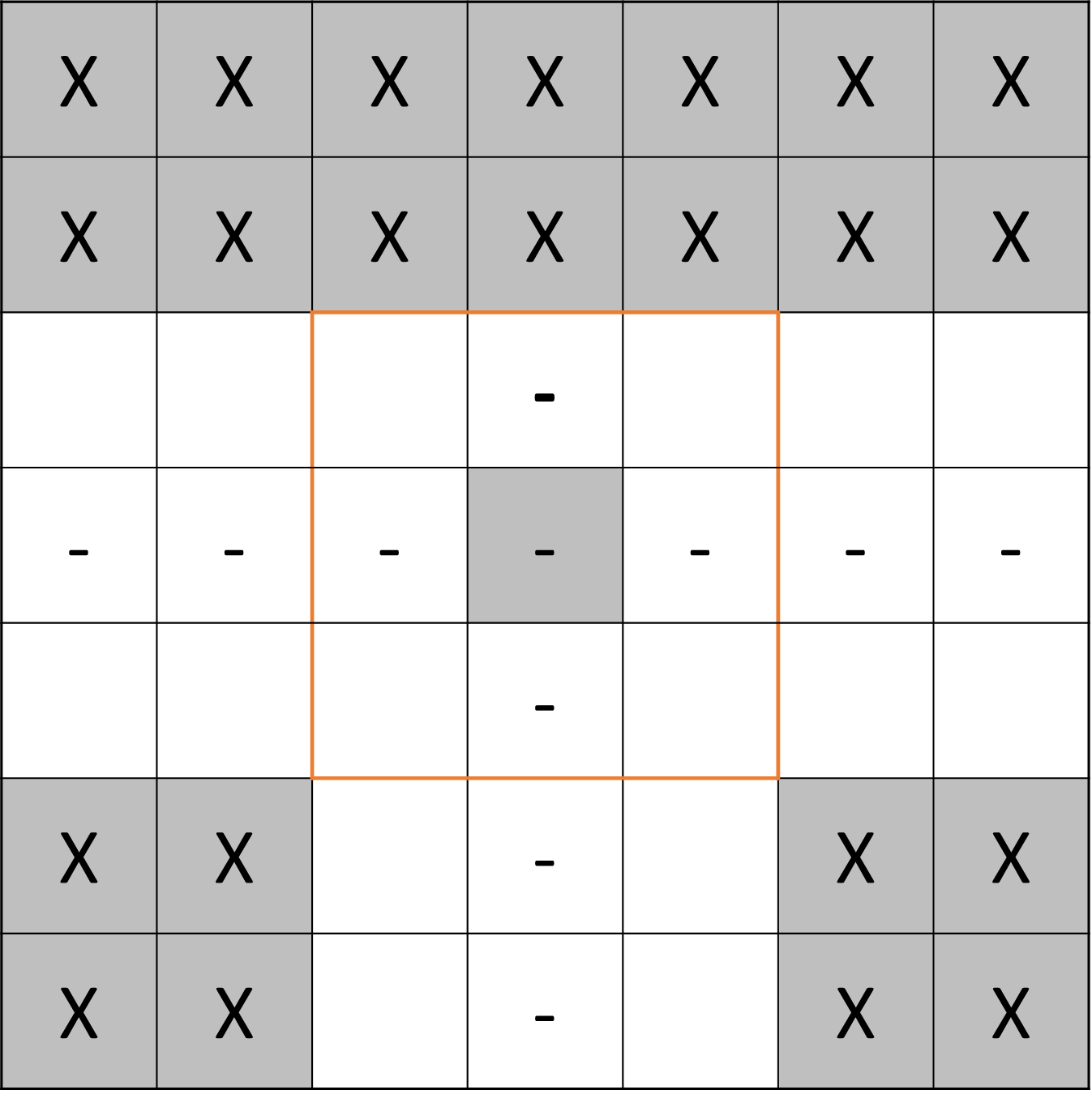
교차로 문제에서는 [그림 1~6]과 같은 **삼거리** **교차로에 접근하는 차량들**이 충돌하지 않고 주행할 수 있도록 제어하는 코드를 구현해야 합니다. 이 문제에서 **모든 차량은 개별적인 스레드로 동작**해야 하며, 각 단위 시간(t=1초)에 한 칸을 전진할 수 있습니다.

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

도로 A를 이용하여 이 교차로에 도착한 자동차는 두 갈래 길 중 하나로 진행할 수 있습니다([그림 1] 참조). 이 교차로에서는 차량이 교차로에 진입하면 경로가 겹치는 다른 차량의 진입이 제한됩니다. 예를 들어, 도로 A에 있는 자동차가 도로 C로 진행한다면 도로 C에 있는 자동차는 도로 A로는 진행이 가능하지만([그림 2] 참조]), 도로 B로는 진행할 수 없습니다([그림 3] 참조]). 마찬가지로 [그림 4]에서처럼 도로 B에 있는 자동차가 도로 A로 진행한다면, 도로 A에 있는 자동차가 도로 B로는 진행이 가능하지만, 도로 C로는 진행할 수 없습니다([그림 5] 참조). 또한 [그림6]과 같이 도로 A에 있는 자동차가 도로 B로 진행을 할 때, 도로 B에 있는 자동차는 도로 C로, 도로 C에 있는 자동차는 도로 A로만 진행이 가능합니다.

교차로를 구현하기 위해 ‘projects/crossroads/mapdata.c’ 코드를 참조하기 바라며, 교차로 문제에서 경로를 정의하기 위해 [그림 7]과 같은 7x7 그리드를 사용합니다. 여러분은 세마포어를 사용하여 교차로에서 자동차의 진입이 정해진 규칙에 따라 진행할 수 있도록 구현해야 하며, 구현을 위해 교재의 Producer/Consumer 문제를 참고하십시오.

입력 데이터의 형식은 연속된 3개의 알파벳을 하나의 단위로 하여 각 알파벳 단위가 ‘:’ 기호로 구분된 문자열입니다. 각 알파벳 단위는 **VehicleID, Start, Dest로 구성**됩니다.여기서 VehicleID는 알파벳 소문자 a-z, Start 및 Dest는 알파벳 대문자 A-C로 표시됩니다. 예를 들어, “aAC:bBA” 문자열을 입력하면 차량 ‘a’는 도로 A에서 도로 C로 진행할 예정이며, 차량 ‘b’는 도로B에서 도로 A로 진행할 예정임을 의미합니다. 각 차량 스레드는 임의의 시점에 생성될 수 있으나, 하나의 차량은 각 단위 시간에 한 칸씩만 전진할 수 있도록 해야 합니다.



[그림 7]

실제 실행 예는 [[링크](https://youtu.be/UuzisXdGEHw)]를 통해 확인할 수 있습니다.

|  |
| --- |
| **[프로젝트 제출물]**  1. 프로젝트 폴더 압축 전, ‘make clean’ 명령어를 통해 빌드 파일들을 모두 제거한 결과  2. 문제(1)의 테스트 방법 및 결과와 문제(2)의 교차로 동기화 방법 및 결과를 설명하는 프로젝트 보고서  3. 프로젝트 결과물(1&2)은 조교에게 이메일(jayhwan91lee@gmail.com)로 제출 |
| **[주의사항]**  1. 기존 파일의 소스 코드를 수정할 경우, 어떻게 수정 되었는지를 설명하는 주석을 반드시 삽입  2. 새로운 파일 추가는 ‘projects/’폴더 내로만 제한됨 |