Operating Systems

Spring, 2022

School of Software, CAU

**Project #2**

**- Page Allocation and Task Scheduling –**

Due: Midnight, June 15th, 2022

**1. 프로젝트 개요**

Project #2의 목적은 Pintos를 이용하여 페이지 할당과 스케줄링 알고리즘을 구현하는데 있다. Pintos는 기본 페이지 할당 알고리즘으로 First Fit을 사용하는데, 관련 코드의 수정 및 구현을 통해 **Buddy System으로 페이지 할당 알고리즘을 교체**하여야 한다. 또한 Pintos는 기본 스케줄러로 Round Robin(RR) 스케줄러를 사용하는데, 여러분은 관련 코드의 수정 및 구현을 통해 RR 스케줄러를 **Multi-level Feedback Queue(MFQ) 스케줄러로 교체**하여야 한다.

Project#2 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 할당**을 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 #2을 위한 프로젝트 아카이브(‘cau15841-pintos-qemu\_v2p2.tgz’)

패키지 설치가 완료되면 아래에서 와 같이 프로젝트 아카이브(‘cau15841-pintos-qemuv2p2.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-qemuv2p2.tgz --------------------- ㉠  # cd pintos\_qemu --------------------------------------------- ㉡  # cd threads ------------------------------------------------- ㉢  # make ------------------------------------------------------- ㉣  # cd build  # ../../utils/pintos pa -------------------------------------- ㉤  # ../../utils/pintos mfq [p1.0]:[p2.1]:[p3.2]:[p4.0]:[p5.3] -- ㉥ |

명령어 “../../utils/pintos pa”에서 첫번째 파라미터는 페이지 할당 테스트 실행을 위해, ‘threads/init.c’에서 ‘projects/pa/patest.h’에 정의된 run\_patest 함수를 호출하여 페이지 할당 테스트를 수행하며, 추가 파라미터는 없다. 자세한 내용은 아래 프로젝트 설명의 **(1) Buddy System Page Allocation**을 참조하기 바란다.

명령어 “../../utils/pintos mfq [p1.0]:[p2.1]:[p3.2]:[p4.0]:[p5.3]”에서 첫 번째 파라미터는 스케줄러 테스트 실행을 위해, ‘threads/init.c’에서 ‘projects/mfq/mfq.h’에 정의된 run\_mfqtest 함수를 호출하여 스케줄러 테스트를 수행하며, 두 번째 파라미터 [p1.0]:[p2.1]:[p3.2]:[p4.0]:[p5.3]은 생성하고자 하는 스레드들의 속성을 나타내는데, 자세한 내용은 아래 프로젝트 설명의 **(2) Multi-level Feedback Queue 스케줄러**를 참조하기 바란다.

**3. 프로젝트 설명**

**(1) Buddy System Page Allocation**

하나 이상의 연속된 메모리 블록을 할당하기 위해 필요한 공간을 효율적으로 탐색하는 것은 페이지 할당자의 중요한 역할 중 하나이다. Pintos의 경우 페이지 할당자로 First Fit 알고리즘을 사용하는데, 여러분은 First Fit 알고리즘 대신 Buddy System 알고리즘을 사용한 페이지 할당자를 구현하여야 한다.

테이블이(가) 표시된 사진

자동 생성된 설명

Buddy System은 메모리를 (, 은 가장 작은 블록의 크기, 은 가장 큰 블록의 크기) 크기의 블록으로 할당하게 되는데, 요청크기 에 가장 가까운 빈공간(이 생성될 때까지 메모리 공간을 절반으로 계속 분할하는 방식이다. 해당 알고리즘에 대한 자세한 설명은 7장 강의를 참고하기 바란다.

또한 여러분은 페이지 상태를 모니터링하기 위한 함수를 구현하여야 한다. 이 함수는 threads/palloc.c 와 threads/palloc.h 에 ***palloc\_get\_status(enum palloc\_flags flags);*** 로 정의되어 있으며, 해당 함수는 **첫번째 페이지부터 마지막 페이지**까지, 각 페이지의 상태( 사용가능 = 0, 사용중 = 1 )를 **한줄에** **32개씩** **출력**하도록 구현하여야 한다.

그리고 여러분이 구현한 페이지 할당자가 잘 동작하는지 확인하기 위해 테스트 방법을 설계하고, 해당 테스트를 수행하는 프로그램을 projects/pa/pa.c에 구현하여야 한다.

**(1.2) 구현 시 요구사항**

1. threads/palloc.h 와 threads/palloc.c 소스코드 내에 Buddy System 알고리즘을 구현하고, Pintos에 이미 구현된 First Fit 알고리즘을 Buddy System 알고리즘으로 교체할 것.

2. threads/palloc.h 와 threads/palloc.c, projects 폴더 내의 파일을 제외한 **나머지 코드는 수정하지 말 것**.

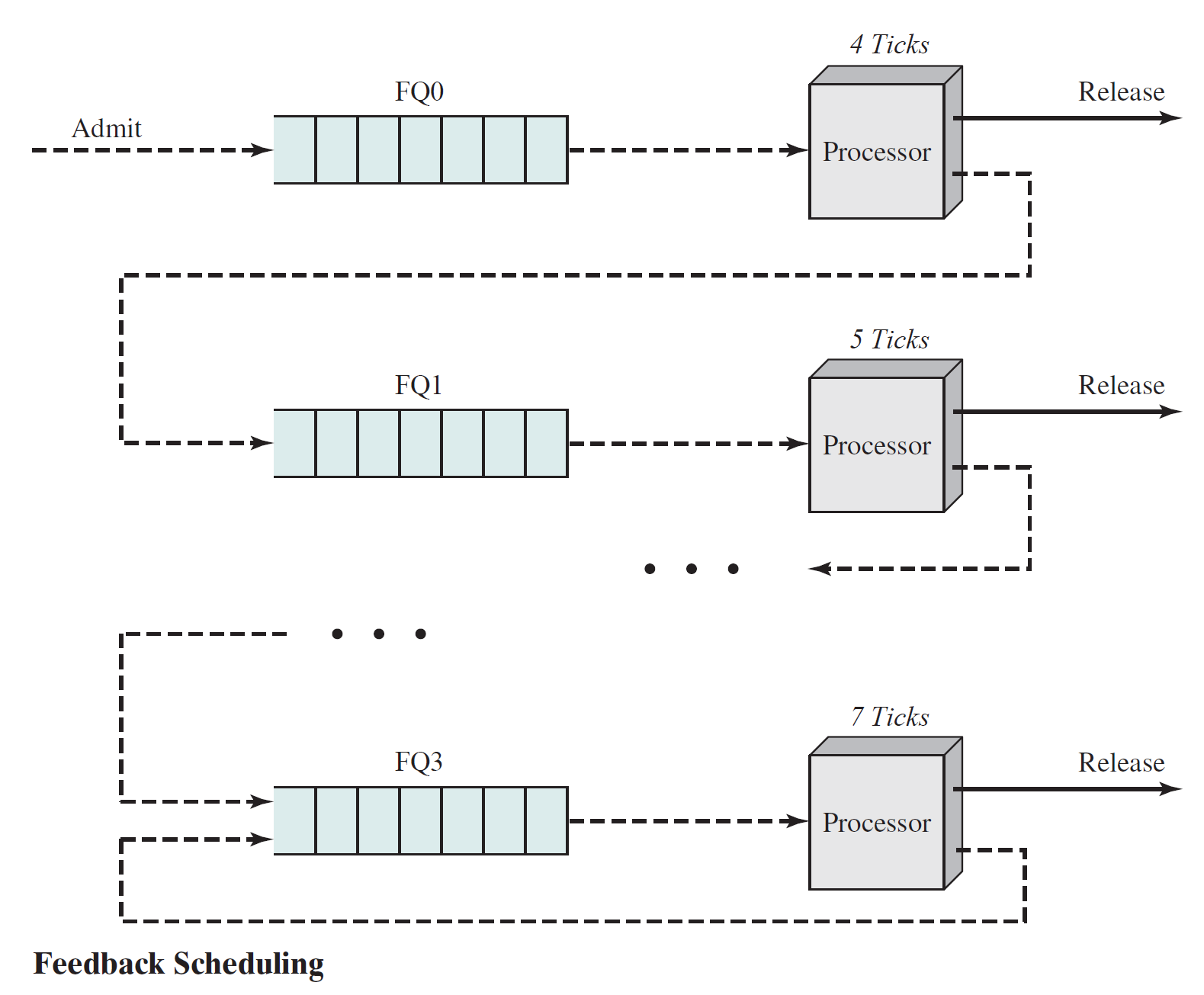
3. 페이지 상태를 모니터링 하기위한 ***palloc\_get\_status*** 함수를 요구사항에 맞게 ‘threads/palloc.c’에 구현할 것.

4. 페이지 할당자의 동작을 확인하기 위한 **테스트 방법을 고안**하여 ‘projects/pa/pa.c’에 구현할 것.

**(2) Multi-level Feedback Queue 스케줄러**

여러 개의 작업을 효과적으로 스케줄링 하는 것이 작업 스케줄러의 목적이다. Pintos의 경우 작업 스케줄러로 Round Robin 알고리즘을 사용하는데, 여러분은 Round Robin 알고리즘 대신 Multi-level Feedback Queue 알고리즘을 사용한 작업 스케줄러를 구현하여야 한다.

여러분이 구현할 MFQ 스케줄러는 4단계 Feedback Dispatcher 방식으로 스레드를 스케줄링 한다. 이 스케줄러의 기본 할당 시간(Time Slice)는 1틱(Tick)이고, Feedback Queue(FQ)인 (Lowest Priority), , , (Highest Priority)의 할당 시간은 각각 7틱, 6틱, 5틱, 4틱이다. 스레드는 생성될 때 할당된 우선순위에 따라 해당 FQ로 이동한다. 예를 들어, 스레드 생성 시 할당된 우선순위가 3(Lowest Priority)인 경우, 이 스레드는 에 배치되며 7틱의 실행 시간을 보장받는다. 4단계 MFQ의 실행과정을 그림으로 설명하면 아래와 같다.



MFQ 스케줄러는 가장 우선순위가 높은 스레드들을 모두 실행시킨 결과 해당 큐가 비게 되면, 다음 우선순위의 FQ에서 대기 중인 스레드들을 실행시킨다. 각 스레드가 실행을 시작한 후, 자신의 할당 시간을 모두 소모하면, 우선순위를 한 단계 낮추어 해당 로 이동한다. 예를 들어, 현재 에 배치된 스레드가 실행을 시작한 후, 이 스레드의 할당 시간인 4틱을 모두 소진하면, 이 스레드의 우선순위는 1로 낮아지고 로 이동한다. 4개의 중에서 가장 높은 우선순위 큐에 있는 모든 스레드들의 실행이 완료되어 큐가 비게 될 경우, 그 다음 높은 (그리고 비어 있지 않은) 큐에 있는 스레드들이 실행된다.

MFQ 스케줄러는 낮은 우선순위 스레드의 Starvation을 막기 위해 Aging 기법을 사용한다. () 에 속한 어떤 스레드 가 실행 중일 때, 이 스레드보다 우선순위가 낮은 큐 () 에 있는 ***실행 준비*** 상태의 스레드 들의 Age는 매 틱 마다 1씩 증가된다. 만일 () 에 속한 어떤 스레드 의 Age가 20에 도달하게 되면, 스레드 의 우선순위가 한 단계 상승하여 상위 우선순위 큐 로 이동되며 동시에 Age값이 0으로 초기화 된다. 단, 세마포어나 뮤텍스와 같이 스레드가 ***block*** 상태로 들어가는 경우 바로 상위 우선순위 큐 로 이동하도록 한다.

MFQ가 정상적으로 동작한다면, 동일한 입력이 주어질 때 동일한 순서로 스레드들이 수행될 것이다. ‘projects/mfq/’에는 MFQ 테스트를 위한 코드가 정의되어 있으며, 여러분은 여러분이 구현한 MFQ를 테스트하기 위해 활용 및 수정할 수 있다. 명령어 “../../utils/pintos mfq [p1.0]:[p2.1]:[p3.2]:[p4.0]:[p5.3]”는 ‘projects/mfq’에 정의된 테스트 코드를 호출하며, 파라미터 [p1.0]:[p2.1]:[p3.2]:[p4.0]:[p5.3]는 우선순위가 각각 0, 1, 2, 0, 3인 스레드 p1, p2, p3, p4, p5를 생성하도록 한다.

**(2.2) 구현 시 요구사항 요약**

1. threads/thread.h 와 threads/thread.c 소스코드 내에 **MFQ 스케줄러를 구현**하고, Pintos에 이미 구현된 RR 스케줄러를 **MFQ 스케줄러로 교체**할 것.

2. threads/thread.h 와 threads/thread.c, projects 폴더 내의 파일을 제외한 **나머지 코드는 수정하지 말 것**.

3. MFQ 스케줄러의 동작을 확인하기 위한 **테스트 방법을 고안**하여 ‘projects/mfq/mfq.c’에 구현할 것.

**(3) 프로젝트 제출물 및 주의사항**

|  |
| --- |
| **[프로젝트 제출물]**   1. 소스코드 폴더 압축 전, ‘make clean’ 명령어를 이용하여 빌드 파일들을 모두 제거한 결과를 압축 2. 소스코드는 pintos\_qemu 폴더를 통째로 압축   3. 페이지 할당자와 스케줄러를 구현한 방법 및 결과를 설명하는 프로젝트 보고서  4. 프로젝트 결과물(소스코드 압축파일 + 프로젝트 보고서)은 압축하여 e-클래스를 통해 제출 |
| **[주의사항]**   1. 새로운 파일 추가는 ‘projects/mfq’, ‘projects/pa’ 폴더 내로만 제한됨. 2. 코드의 수정은 ‘projects’ 폴더 내부의 파일, ‘threads’ 폴더 내부 ‘palloc.c’, ‘palloc.h’, ‘thread.c’, ‘thread.h’ 로 제한함. 3. 압축시 egg로 압축하지 말것 |