PintOS in UOS

운영체제 프로젝트 발표

2기 박주영
2기 나윤서
3기 허준재

PintOS란?

스탠포드에서 개발한 운영체제 프레임워크입니다.
학생들이 직접 OS의 핵심 기능을 구현하며 운영체제의 내부 동작원리를 깊게 이해할 수 있도록 설계되었음
단순한 이론 학습을 넘어, 스레드, 메모리 관리, 시스템 콜 등 추상적인 개념을 코드로 구현할 수 있음

PintOs 커리큘럼

1. Thread
2. User Programs
3. Virtual Memory
4. File Sysstem

PROJECT 1 – THREADS

- Alarm Clock
- Priority Scheduling
- Advanced Scheduler (Extra)

PROJECT 2 – USER PROGRAMS

- Argument Passing
- User Memory Access
- System Calls
- Process Termination Message
- Deny Write on Executables
- Extend File Descriptor (Extra)

Thread(Alarm clock)

과제 목표

timer_sleep()함수를 구현하여, 지정된 시간(tick)동안 현재 스레드의 실행을 멈추고 다른 스레드에게 CPU를 양보하도록 만들기

핵심개념

타이머 인터럽트

일정시간마다 주기적으로 발생하는 하드웨어 인터럽트

thread 상태관리

Running,Ready, Blocked 상태 전이

구현전략

Sleeping Queue 도입

Sleep 스레드들을 관리하기 위한 별도의 큐를 생성한다.

timer_sleep()로직

-현재 스레드를 BLOCKED 상태로 변경
-깨어날 시간을 스레드 구조체에 저장
-스레드를 Sleeping Queue에 삽입 후 다른 스레드 실행

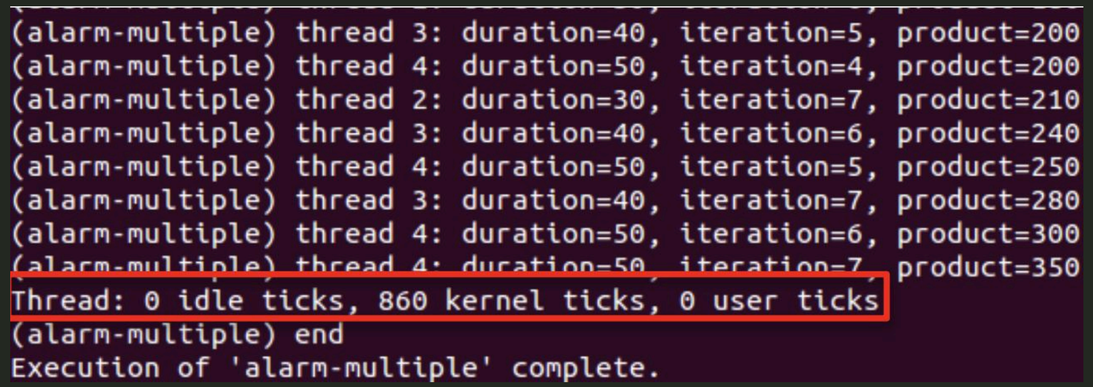
timer_interrupt()수정

-타이머 인터럽트마다 Sleeping Queue순회
-깨어날 시간이 된 스레드는 READY상태로 변경

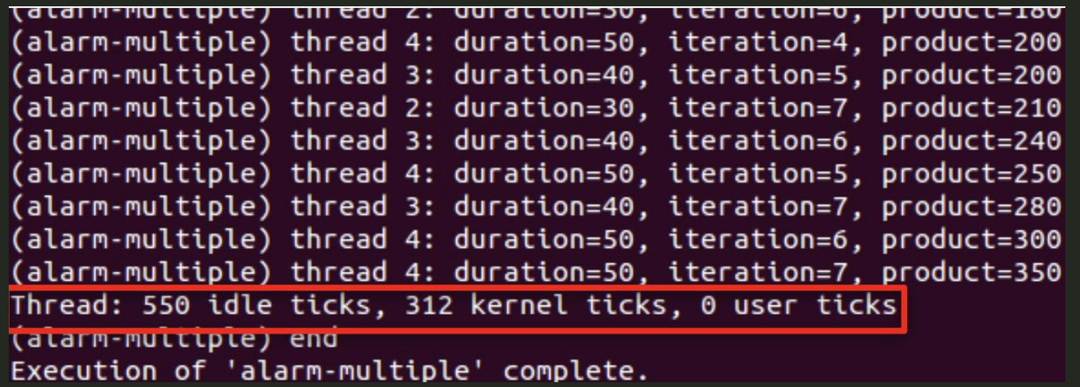
Thread(Alarm clock)

결과

수정 전



수정 후



sleep queue를 이용하면 스레드는 기다릴 때 CPU를 점유하지 않고 잠들기 때문에, 커널 틱과 전체 사용률이 줄고 idle 틱 늘어난다. 즉, CPU 효율이 좋아졌다!

Priority Scheduling

과제 목표

스레드에 우선순위를 부여하고 스케줄러가 항상 가장 높은 우선순위의 스레드를 실행하도록 하는 것

핵심개념

선점형 스케줄링

현재 실행중인 스레드보다 높은 우선순위 스레드가 등장하면 즉시 CPU를 빼앗는 방식

우선순위 역전

높은 우선순위 스레드가 낮은 우선순위가 보유한 자원을 기다리는 상황

구현전략

Ready Queue 정렬

우선순위가 높은 순서로 Ready Queue 삽입 로직 변경

Preemption 구현

우선순위 변경/스레드 추가 시 즉시 양보 결정

Priority Donation

우선순위 역전 방지를 위한 임시 우선순위 상향

Priority Scheduling

결과

수정 전

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs/mlfqs-load-1
FAIL tests/threads/mlfqs/mlfqs-load-60
FAIL tests/threads/mlfqs/mlfqs-load-avg
FAIL tests/threads/mlfqs/mlfqs-recent-1
pass tests/threads/mlfqs/mlfqs-fair-2
pass tests/threads/mlfqs/mlfqs-fair-20
FAIL tests/threads/mlfqs/mlfqs-nice-2
FAIL tests/threads/mlfqs/mlfqs-nice-10
FAIL tests/threads/mlfqs/mlfqs-block
7 of 27 tests failed.
```

수정 후

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs/mlfqs-load-1
FAIL tests/threads/mlfqs/mlfqs-load-60
FAIL tests/threads/mlfqs/mlfqs-load-avg
FAIL tests/threads/mlfqs/mlfqs-recent-1
pass tests/threads/mlfqs/mlfqs-fair-2
pass tests/threads/mlfqs/mlfqs-fair-20
FAIL tests/threads/mlfqs/mlfqs-nice-2
FAIL tests/threads/mlfqs/mlfqs-nice-10
FAIL tests/threads/mlfqs/mlfqs-block
7 of 27 tests failed.
```

priority scheduling과 priority donation 관련 테스트 성공!

Argument Passing

과제 목표

ls -l과 같이 사용자가 입력한 명령어 문자열 파싱하며, 새로 생성되는 프로세스의 스택에 인자(argc,argv)를 올바르게 전달하는 것이다.

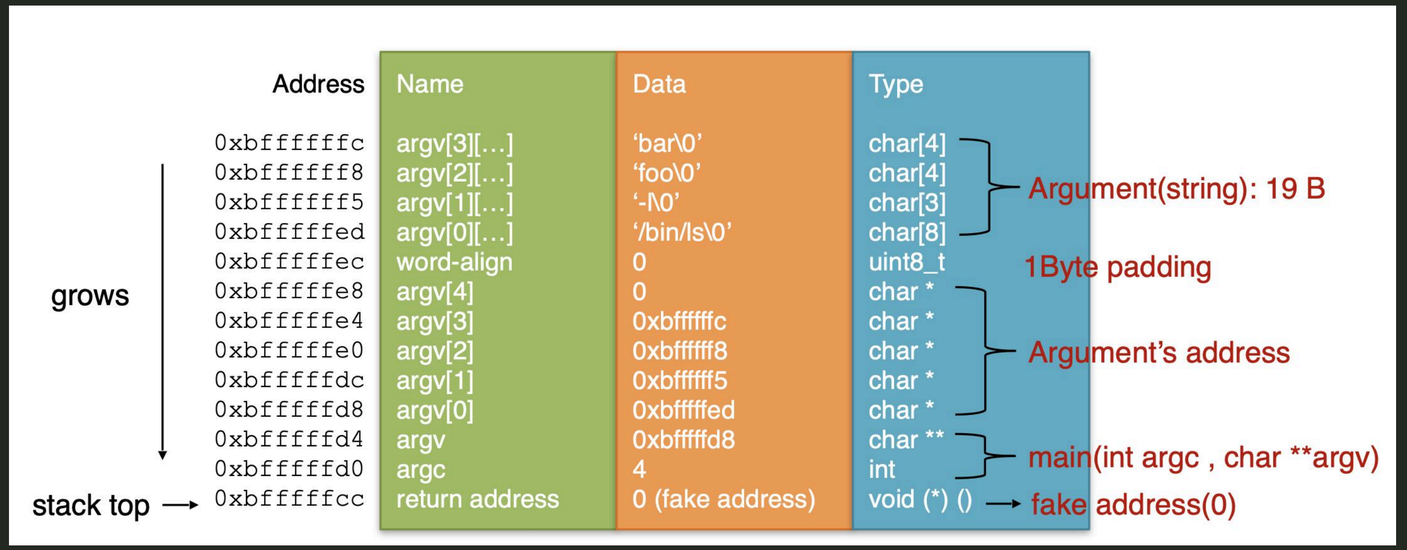
핵심개념

문자열 파싱

strtok_r함수를 사용하여 입력된 명령어 문자열을 공백 기준으로 토큰(인자)들로 분리합니다.

스택 구성(역순)

파싱된 인자 문자열들을 스택의 가장 높은 주소부터 차례로 push하고, Wold-align을 위해 패딩 바이트를 추가합니다. 이후 문자열을 가리키는 포인터 주소들, argv 배열의 시작주소, argc, 반환 주소를 push 합니다.



User Memory Access

과제 목표

시스템 콜 핸들러에서 사용자 프로그램이 전달한 포인터가 유효한지 검증하는 안정장치를 마련하는 것

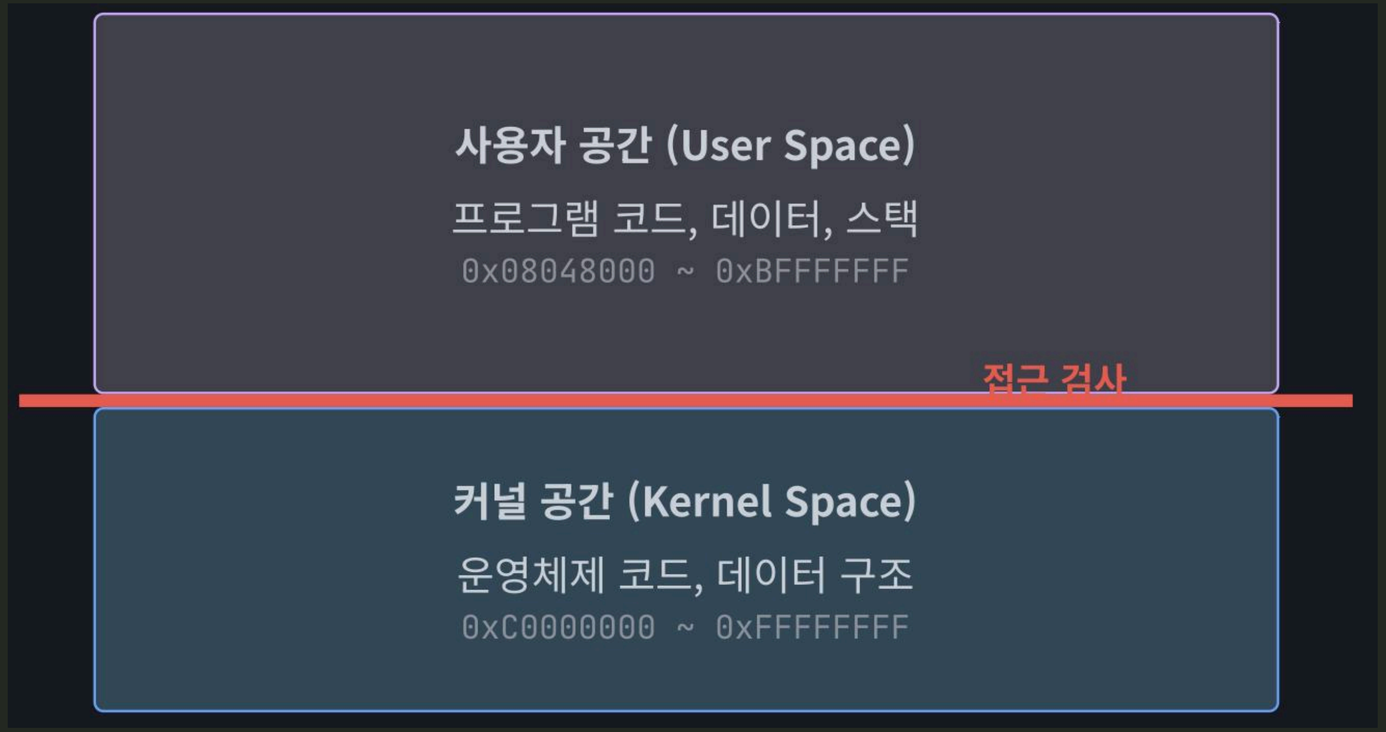
핵심개념

메모리 보호

커널 영역을 사용자 프로그램으로부터 보호해
시스템 안정성 유지

주소 공간 분리

커널 공간과 사용자 공간을 명확히 구분하여
접근 제어



System Call

과제 목표

fork,exec,wait,open,read,write 등 필수적인 시스템 콜들을 구현하여 사용자 프로그램이 운영체제의 기능을 활용할 수 있도록 합니다.

핵심개념

시스템 콜 핸들러

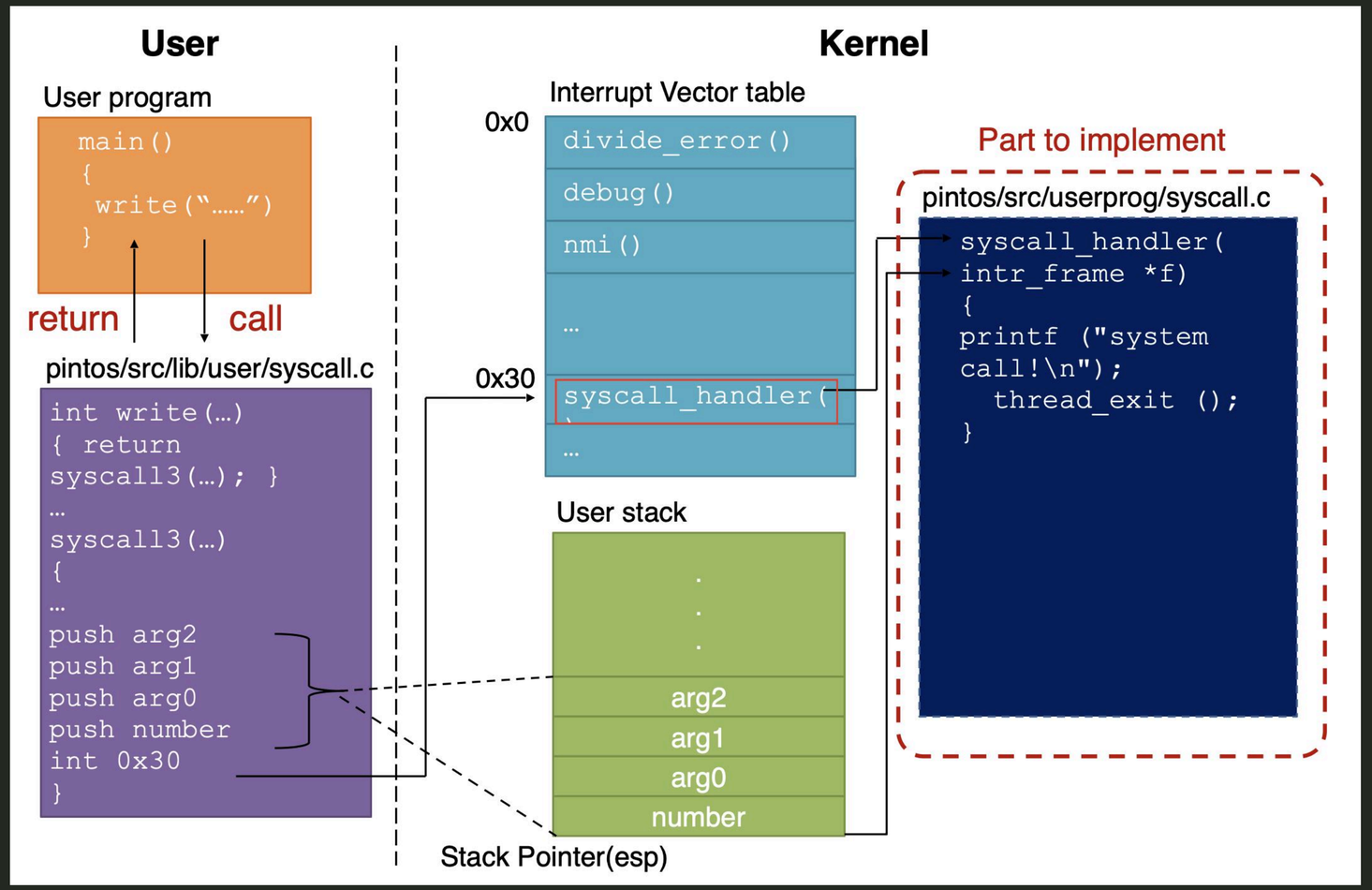
사용자 프로그램의 요청을 받아 커널 내부 기능으로 연결

인터럽트 벡터

특정 인터럽트 번호와 처리 루틴을 연결

파일 디스크립터

사용자 프로세스와 시스템 파일리소스를 연결하는 추상화



System Call

구현 전략

시스템 콜 핸들러 설정

인터럽트 벡터 테이블에 시스템 콜을 위한 엔트리를 등록하고, 해당 인터럽트 발생 시 홍출될 핸들러 함수 연결

argument 추출

핸들러 함수 내에서 사용자 스택에 저장된 시스템 콜 번호와 인자들을 읽어옴

기능 분기

읽어온 시스템 콜 번호에 따라 switch-case문을 사용하여 각각의 기능을 수행하는 커널함수를 호출(sys_exec, sys_wait)

파일디스크립터 테이블

각 프로세스마다 파일 디스크립터 테이블을 갖도록 PCB(process control block)를 확장하여 open, read, write시 파일 객체를 관리하도록 구현

공부방식

이론선행

1. 운영체제 공룡책 정독
2. pintOs_kaist 문서 정독
3. 주요개념에 대한 지식 공유 진행

기록

블로그에 공부한 내용과 구현과정을 기록

역할분담 및 협업

1. 프로젝트 단위별로 각 팀원이 맡아 구현 후 설명
2. github pull request를 통한 코드리뷰 진행

힘들었던 점

버전문제

VSCode가 Ubuntu 18.04 지원 중단 → PintOS 실행 환경 충돌 발생
Ubuntu 20.04/22.04 + Docker 조합 시 디버깅 오류 지속
최종적으로 SFTP 방식으로 VSCode ↔ Terminal 연동
비효율적 개발 과정으로 시간 소모가 컸음

느낀점

학습자료의 필요성

강의에서 진행하는 수업이기도 하고, 버전문제 및 설치방법등 자료가 부족해서 스터디하는데
불필요한 시간을 많이 소모하였음.
충분한 학습자료가 제공되는 커리큘럼을 고르는 것이 중요하다.

실습의 중요성

운영체제를 개념으로만 배우다가, 실제로 구현해보니 무작정 암기하는 것보다는 재밌게 공부할 수 있었다.

감사합니다