[크래프톤 정글] PintOS Project

기간	@2022년 12월 16일 → 2023년 2월 2일
태그	C Github OS Team Project

Overview

PintOS는 스탠포드 대학교 컴퓨터 공학과 수업 'CS 140'을 위해 개발된 교육용 mini OS입니다.

x86_64 운영체제에서 PintOS를 개발하기 위해 KAIST에서 개발한 PintOS-kaist 기반으로 프로젝트를 진행하였으며 Project는 총 4단계로 나누어져 있습니다.

각 단계 별로 C언어를 사용하여 mini OS의 필수 부분을 구현하며 컴퓨터 시스템의 중요한 부분을 이해하고 디버깅 기술을 학습하였습니다.

🙄Goal

- 3인 1조로 PintOS Project를 수행하여 OS 동작 원리에 대해 학습 (5주, 3주차에 팀원 변경)
- PintOS 프로젝트 구현
  1. alarm clock(완료), thread(완료), mlfqs(optional)
  2. system call (완료)
  3. page fault handler(완료), mmap, munmap, (미진행)
  4. file system (미진행)

Stack

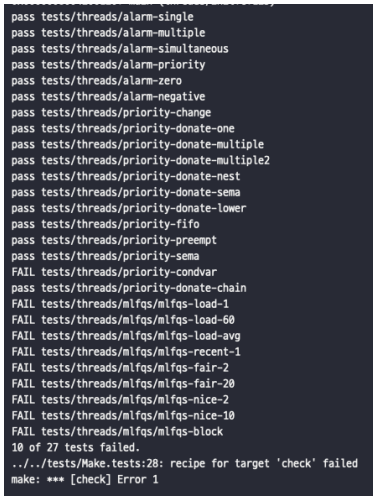
- 언어 : C
- Tools : Github, Live Share(Pair Programming)

Role

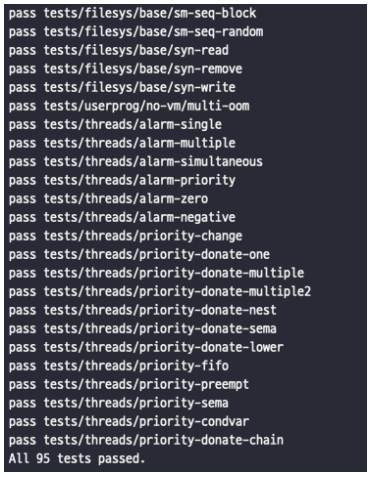
- PintOS 관련 이론 학습 (https://premise.oopy.io/proejct/pintos/personal)
- C를 사용한 PintOS Project 구현
- 문제 해결 및 테스트

결과 화면

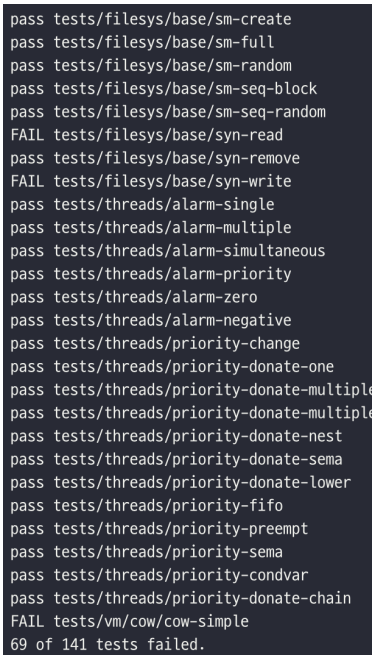
- 각 프로젝트 테스트 결과화면



Project1 (Sleep & Wake-up and Priority Donation 구현)



Project2 (Exit, Wait System Call 개선 포함)



Project3 (Virtual Memory Lazy Loading 까지 구현)

어려웠던 점

가장 어려웠던 점은 Project2의 System Call 구현 중 wait과 exit 코드 개선 부분이었습니다.

다른 팀으로부터 전달받은 wait과 exit 코드는 테스트케이스는 통과하지만, 부모에서 wait을 호출하지 않고 자식이 exit 하는 경우에서 무한 loop에 빠져 메모리 leak이 발생하는 문제점이 있었습니다.

이 부분을 개선하기 위해 새로운 팀원들과 4일동안 문제를 해결하기 위해 노력하였고 개선을 위해 추가 작성한 코드로만 시야가 좁아져 문제 해결에 어려움이 있었습니다. 문제 해결의 실마리를 찾기 위해 그동안 테스트한 내용과 작성한 코드를 바탕으로 과거 코드를 작성한 동기에게 코드 리뷰를 요청하였습니다. 코드 리뷰를 통해 기존 코드에 exit 호출 후 ready-list에서 종료 되지 않은 쓰레드가 먼저 지워지는 문제가 있던 것을 확인하고 해당 코드를 제거하여 문제를 해결하였습니다.

이 경험을 통해 테스트를 통과했더라도 문제가 발생 할 수 있다는 것을 크게 느꼈습니다.

Takeaway

Project1 (Thread)

- 코드의 많은 곳을 변경해야 하므로 어디를 어떻게 변경했는지 기록을 관리하는 게 중요하다고 느꼈습니다.

  또한, 백업을 습관화하고 이런 코드가 방대한 프로그램의 경우 vscode extension의 “TODO” 같은 기능이나 함수 선언들 들어가고 나가는 여러 유용한 단축키를 잘 활용하는 것이 시간 줄이기 위한 팁임을 알게 되었습니다.

- 코드를 잘 분석하는 것도 개발자에게 중요한 능력이라고 느끼게 되었습니다.

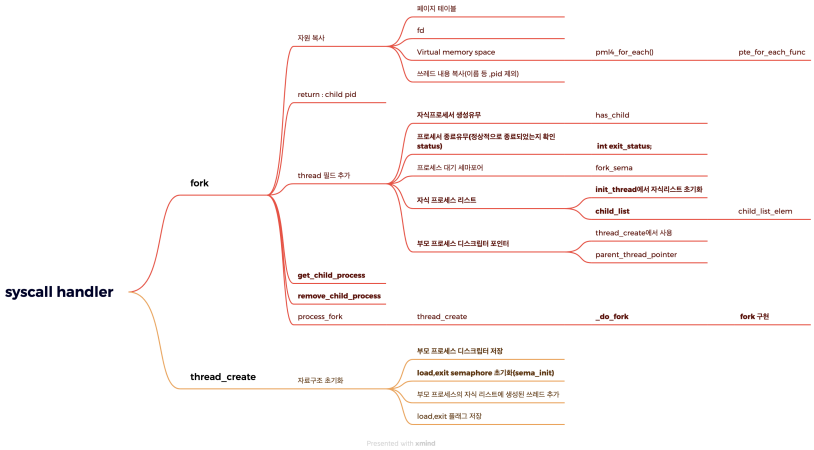
- “simple is the best” 라는 말이 생각이 떠올랐습니다.

  OS 내부에서 작동하는 함수가 생각보다 엄청나게 복잡하지 않았고, 딱 해야 하는 동작하도록 구현해야 했습니다. 코드가 방대하고 스레드와 lock이 어떤 식으로 동작하는지 파악하는 것이 헷갈리고 까다로웠지만 코드 C언어의 문법적인 어려움은 `&` 을 써야 할지 말아야 할지, `*`을 써야 할지 말아야 할지 정도였습니다.

이 경험을 통해 함수는 작고 한가지 동작만 하게 만드는 편이 좋은 함수라는 것을 다시 한번 느끼게 되었습니다.

Project2 (System Call)

- 어디서부터 뭘 해야 할지 모르겠다면 어떻게 흘러갈지 흐름부터 잡는 것이 도움이 되었습니다.
    - 시스템 콜이 호출되는 과정을 먼저 잡으니 어디에 무엇을 구현할지가 조금씩 감이 잡혔고, 시스템 콜 하나를 구현하기 위해 필요한 것이 무엇인지 분류를 하니 어떤 식으로 접근해야 할지 명확 해졌습니다.



- 급하게 하지 말고 하나씩 정확히 하는 게 좋다는 것을 느꼈습니다.
    - fork에서 Sema를 이상하게 거니까 close가 안되고 하나를 고치니 다른 곳에서 에러가 발생하였습니다. 구현해야 할 양이 많고 이게 맞는지 아닌지 잘 모르는 어려운 상황임에도 불구하고 적어도 내가 작성한 코드는 정확히 이해해야 나중에 터졌을 때 금방 해결이 되었습니다. 💩

Project3 (System call exit & wait 수정, Virtual Memory)

- 개선한 System call exit & wait 부분은 어디서 문제가 발생했는지 추가한 코드와 변경한 코드 위주로 파악하였습니다. 정작 문제는 다른 부분에서 문제가 발생했지만, 제가 어떤 점이 문제인 건지 제대로 인식하지 못하고 무작정 디버깅만을 진행하여 문제점을 찾는데 시간이 너무 오래 걸렸습니다.
- 이 경험을 통해 배운 점은 지금 당장 잘 작동되는 코드라도 잠재적인 문제가 있을 수 있음을 배웠습니다. 문제가 생기면 다른 부분은 잘될 거라는 안일한 생각을 하지 말고 의심하고 직접 확인해야한다는 경험을 하였습니다.

Etc

- 개인 개발 일지 : https://premise.oopy.io/proejct/pintos/personal