**Pintos Project 2**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070]운영체제

담당 교수 : 소정민 교수님

조 / 조원 : 41조 20130859 송민규, 20130956 장동욱

개발 기간 : 2018.09.22~10.18

**프로젝트 제목 : pintos project 2**

**제출일 : 2018.11.14**

**참여 조원 : 20130859 송민규, 20130956 장동욱**

1. **개발 목표**

* **filesystem 관련 system call 구현**
* **lock, semaphore 구조체를 활용한 synchronization 구현**
* **project1에 구현된 wait를 semaphore로 대체**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **System Call**

**file system을 활용하는 system call create, remove, open, close, filesize, read, write, seek, tell을 구현한다. create, remove는 file을 생성하고 삭제한다. open은 file에 대한 file descriptor를 생성하고 열고, close는 파일을 닫는다. 나머지 함수는 file descriptor를 인자로 하지만 filesystem 함수의 인자는 file 구조체이기 때문에, file descriptor를 file 구조체로 변환해주는 fd\_to\_file함수를 구현한다. read와 write는 인자 fd에 상응하는 파일을 읽고 쓴다. seek은 파일 내 커서를 옮기는 기능을 수행하고 tell은 현재 커서를 반환한다. 커서는 file내 다음으로 읽거나 쓸 byte를 가리킨다.**

* **Lock**

**manual의 요구에 맞추어 filesystem 함수에 대한 동시 접근을 방지하고 사용한다. 본 프로젝트의 구현 대상인 system call의 예외 처리 부분 뒤에 lock\_acquire를 추가하고, 함수 마지막에 lock\_release를 추가한다. lock으로 둘러싸인 critical section 안에서 예외처리가 발생한다면 return 전에 lock\_release를 추가하여 lock이 묶여있는 상태를 방지한다.**

* **Semaphore**
  + 1. **status lock**

**부모 프로세스는 wait 수행 결과로 자식 프로세스의 exit status를 반환해야 한다. wait가 수행되기 전에 자식 프로세스가 exit 함수에서 인자로 받은 status를 자신의 thread 구조체 필드인 exit\_status에 할당할 수 있도록 기다려준다. process\_wait에서 wait에 대한 예외 처리가 끝난 후 sema\_down을 호출하고, exit에서 exit\_status를 할당받은 후에 sema\_up을 호출한다.**

* + 1. **exit lock**

**이후 부모 프로세스가 process\_wait에서 자식의 exit\_status collect하기 전 자식이 thread\_exit해버리면 collect가 불가능하다. 따라서 자식 프로세스가 exit 함수에서 부모함수가 collect할 때까지 기다려준다. exit에서 status lock에 대한 sema\_up이 끝난 다음 바로 sema\_down하고, process\_wait에서 collect가 끝난 후 sema\_up한다.**

* + 1. **remove lock**

**본 프로젝트에서는 프로세스가 현재 수행 중인지 아닌지를 판단하는 수단으로서 all\_list에 해당 프로세스가 존재하는지의 여부를 사용한다. 따라서 부모 프로세스는 wait에서 자식 프로세스가 all\_list에서 list\_remove되는 순간까지 기다려야만 프로세스가 현재 수행 중인지 판별하는 코드들이 의도대로 수행된다. wait에서 exit lock에 대한 sema\_up이 끝난 다음 sema\_down하고, thread\_exit에서 list\_remove 다음에 sema\_up한다.**

* 1. **개발 내용**
* **file\_deny\_write, file\_allow\_write**

**rox 테스트 케이스를 통과하기 위해서는 실행파일에 대한 write를 금지시켜야 한다. 이를 위해 file open시 file 구조체 필드인 name에 파일명을 할당하고, file 구조체를 열람자 thread의 구조체 필드 file\_list에 list\_push한다. file을 write하기 전 해당 file의 이름과 all\_list 상의 현재 수행중인 프로세스의 이름과 비교하여 같다면 file\_deny\_write한다.**

* **close\_all\_files**

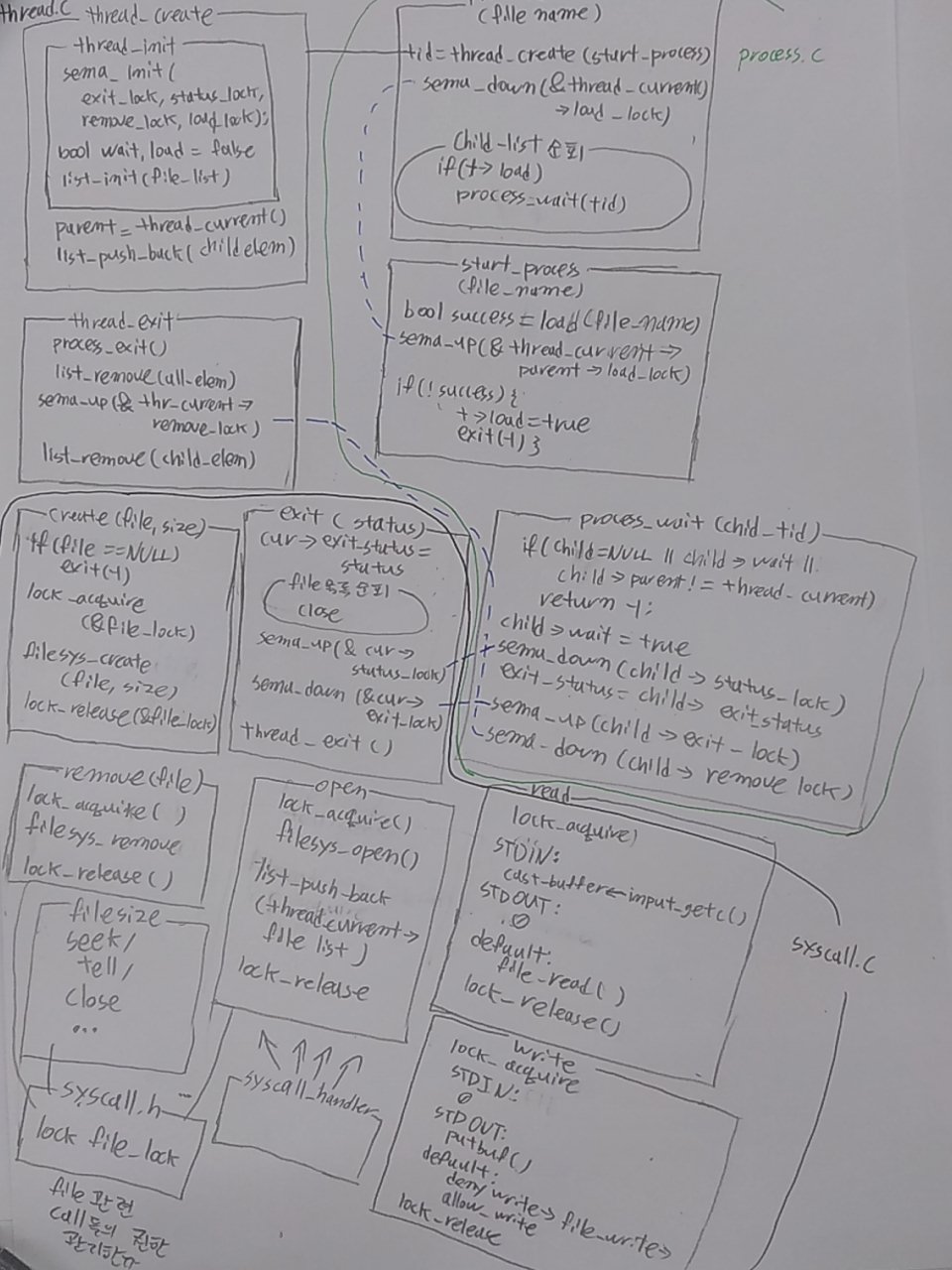
**multi-oom 테스트 케이스에서 생성된 n>=16인 프로세스들은 consume함수를 통해 open을 126번 수행한다. 해당 케이스를 통과하기 위해서 프로세스가 종료되기 전 thread의 구조체 필드인 file\_list에 추가된 모든 file을 close해야 한다. 이를 위해 close\_all\_file함수를 구현한다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **10.25–10.31명세서 탐독 및 자료조사**
* **1.1-11.7 코드 작성**
* **11.8-11.14 디버깅 및 보고서 작성**
* **11.14 사이버캠퍼스 제출**
  1. **개발 방법**
* **코드 구현과 수정은 vim으로 수행했다. 디버깅은 hex\_dump와 출력문을 활용했다. 테스트와 디버깅은 make test, make grade, backtrace 명령문과 output파일, result파일을 활용했다.**
  1. **연구원 역할 분담**
* **semaphore부분과 filesystem 관련 system부분으로 나누어 각자 기본적인 구현을 진행하고, 테스트 및 디버깅을 함께 수행했다.**

1. **연구 결과**
   1. **합성 내용**

**프로젝트 구현사항을 구조도로 나타내면 다음과 같다.**

****

* 1. **제작 내용**
* **syscall.c, syscall.h, file.c, file.h, thread.c, thread.h, process.c를 수정하여 개발 내용의 내용을 구현하였다.**
  1. **시험 및 평가 내용**

**명세서에 주어진 모든 요구사항을 만족하고, make check 및 make grade 결과 모든 테스트 케이스를 통과했다.**

1. **기타**
   1. **연구 조원 기여도**

* **송민규(50%), 장동욱(50%)**
  1. **소감**
* **프로그램의 여러 부분이 복잡하게 상호작용을 하여, 디버깅이 어렵고 수정 사항의 연쇄작용을 파악하는 것이 어려웠다. 테스트 케이스가 많아 테스트 시간이 오래 걸려 전체 프로젝트 소요 시간이 길어졌다.**