**Pintos 프로젝트 2\_2. Pintos User Program**

**(설계 프로젝트 수행 결과)**

**과목명 : [CSE4070-01] 운영체제**

**담당교수 : 서강대학교 컴퓨터공학과 박성용**

**조원 : 11조 권나영 박상욱**

**개발기간 : 2016. 11. 10. -2016. 11. 15.**

**최 종 보 고 서**

**프로젝트 제목: Pintos 프로젝트 2\_2. Pintos User Program**

**제출일: 2016. 11. 18 .**

**참여조원: 11조 권나영 박상욱**

**I. 개발 목표**

Project 2\_1에서 못다한 filesystem 관련 systemcall을 구현하는 것이 이번 프로젝트의 가장 중요한 목표이다. system call (create, remove, open, close, filesize, read, write, seek, tell)이 잘 수행될 수 있도록 만들어준다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

1) project2-1에서 busywaiting방식 때문에 났던 error들을 semaphore 개념을 도입하여 wait, exec를 수정한다. pintos/src/lib/user.c에 존재하는 assembly code를 수정하여 systemcall을 수행할 때 argument들이 밀려서 들어오는 것을 방지한다.

2) 이번 프로젝트에서 구현해야 할 System call(create, remove, open, close, filesize, read, write, seek, tell)을 구현하고 user program 작동시 synchronization 문제를 해결한다. test case의 결과에 따라 fail이 뜨는 곳들의 error를 고친다.

**나. 개발 내용**

1) Synchronaization

세 개의 semaphore를 사용해서, proj2\_1에서 busy-waiting으로 제대로 동작하지 않았던 synchoronaization을 수정하고, process\_exec,process\_wait,process\_exit이 의도한 방향으로 잘 작동하게 수정하였다. sema\_wait, sema\_exit, sema\_elim을이용해서 thread의 Synchoronaization을 수행하였다.

또한 PintOs의 File System을 구현하기 위해서는 Lock을 사용하였다. File System의 System call을 사용하는 부분(예: open, read, write )에서, lock\_acquaire, lock\_release를 사용해 Synchronaization을 구현하였다.

2) file system을 구현하기 위해 struct thread에 file\_descriptor의 개수와 file descriptor table, 현재 수행되고있는 파일인 exec\_file을 추가한다. pintos는 실행 중인 프로그램의 executable file에 다시 쓰기를 원하지 않기 때문에 load 함수에서 현재 실행되고 있는 파일에 쓰지 못하도록 file\_deny\_write함수를 호출한다. 이는 file\_close할 때 그 함수안에서 file\_allow\_write()함수를 호출하기 때문에 다시 write가 가능해진다.

syscall\_create : 파일의 이름과 초기화할 크기가 주어지면, 파일을 생성한다,

syscall\_remove : 주어진 이름의 파일을 삭제한다

syscall\_open : 주어진 이름의 파일을 이용하여 file을 열고 현재 thread의 file discriptor에 연결하고 그 file descriptor 번호를 반환한다.

syscall\_close : 주어진 이름의 파일을 닫아준다

syscall\_filesize : 주어진 파일의 사이즈를 반환한다

syscall\_read : 해당하는 fd에 존재하는 file을 열어, 버퍼에 저장된 문자열을 size byte만큼 읽어 저장한다

syscall\_write : 해당하는 fd에 존재하는 file을 열어,sizebyte만큼 버퍼의 문자열을 출력한다.

syscall\_seek : 현재 파일에서 읽거나 쓸 위치를 반환한다

syscall\_tell : 파일의 현재 읽거나 쓰일 위치를 반환한다.

프로젝트 2\_1과 마찬가지로, System call 함수들의 return값은 (intr\_frame \*f)의 eax register에 저장해준다. 이 때 esp로 접근하는 argument들의 주소가 physical base를 넘지 않는지 확인해야 한다.

systemcall을 구현하면서 make check를 통해 fail 떴던 test들에 대해 예외처리를 한다. 총 76개 중에 create-null, read-bad-ptr, bad-read, multi-oom의 경우에 fail이 나왔다. 알맞은 예외처리를 하여 Exception을 대비한다.(방법은 아래에 서술)

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

- 11월 10일 ~ 11월 12일 : project2\_1에서 발생한 wait, exit exec 관련 오류들을 분석하여 synchronization을 busywaiting방식에서 semaphore를 이용하는 방식으로 바꿨다. 이번 프로젝트에 사용될 struct에 대해 분석하고 추가될 system call에 대해 전반적인 흐름을 연구했다.

- 11월 13일 ~ 15일 : system call 추가되는 함수 구현, 오류를 정정하였다.

- 11월 16일 : 내구성과 안전성을 위해 10 번의 make check를 통해 모든 테스트케이스에 대해 통과되는지 확인해보고 보고서를 작성하였다.

**나. 개발 방법**

csprp1과 cspro2 서버에서 vi editor를 가지고 강의자료와 pintos 매뉴얼을 참고하여 개발한다. 개발 환경은 linux 컴파일 환경은 GCC이고, make file을 수정하여 project2\_1과는 다른 linker를 사용했다.

**다. 연구원 역할 분담**

권나영 : 매뉴얼 분석, proj#2-2 ppt자료 분석, system call 구현, 보고서작성

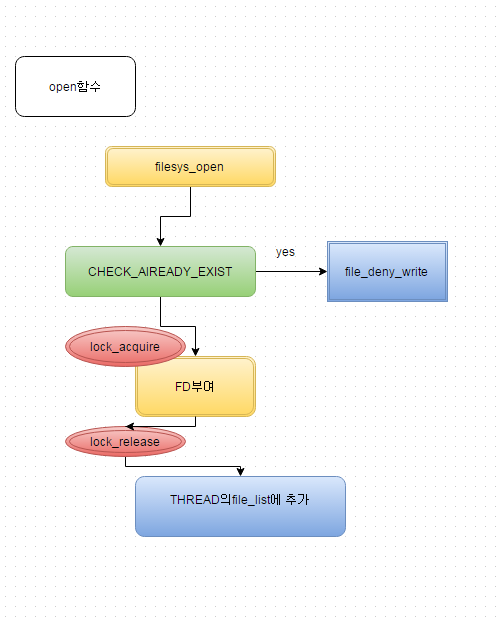
박상욱 : 매뉴얼 분석, proj#2-2 ppt자료 분석, busy waiting방식 semaphore로 바꿔 project2\_1의 오류 고치기, system call 구현, 보고서작성

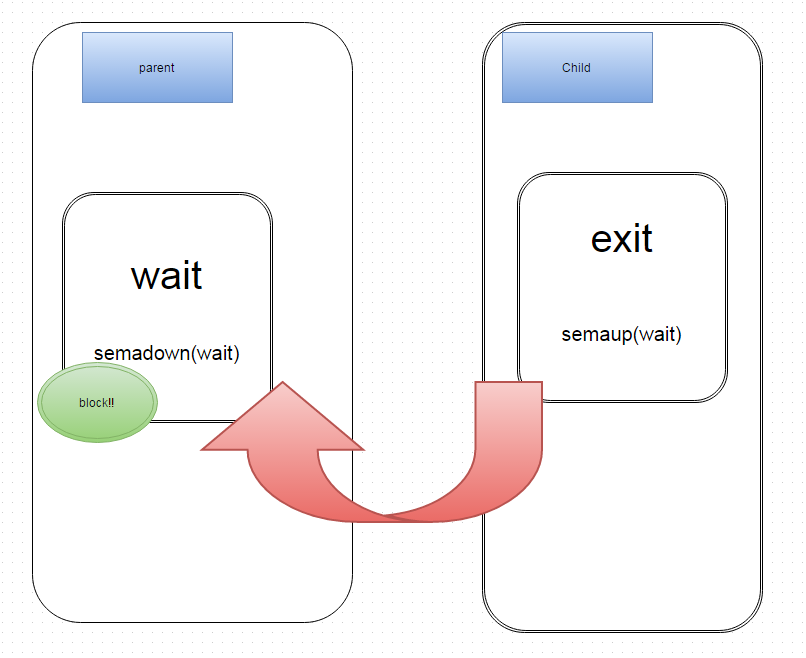
**IV. 연구 결과**

**1. 합성 내용:**

**- 설계 목표에 필요한 내용을 조사 분석한 후 그들을 바탕으로 구성한 전체 프로그램 구성도**

file systemcall API의 동작 과정 :user program level에서 fopen()함수가 호출된다. fopen함수가 내부적으로 open()함수(syscall\_open())를 부르고 syscall\_open이 내부적으로 filesys\_open함수를 부른다. Open된 file을 이용한 작업이 다 끝나면 syscall\_close함수가 호출되고 이 함수 안에서 file\_close함수가 불려지고 file descriptor를 비롯한 변수들을 사용 해제해준다. 프로세스가 끝나기 전에 열린 file에 대해서 file\_close함수를 이용하여 다 닫아준다.





**2. 제작 내용:** 개발 결과

- 구현한 컴포넌트들에 대한 역할 및 구현 방법에 관한 내용을 기술할 것. (예: 자료구조, 알고리즘).

**void lock\_init(struct lock \*);**

=>lock를 건다

**void lock\_acquire(struct lock \*);**

=>sleep상태로 만든다 (주로 interrupt handler에서 interrupt disable과 함께 호출)

**void lock\_release(struct lock \*);**

현재 thread에게 소유되고있는 lock을 sema\_up을 통해

**void sema\_init(struct semaphore \*, unsigned value);**

=> semaphore의 list를 초기화하고value값으로 초기값을 지정

**void sema\_down(struct semaphore \*);**

**=>** sema값을 down 시키고 list의 waiter list에 현재 thread를 추가함

**void sema\_up(struct semaphore \*);**

**=>** sema를 incre시키고, waiting list에 있는 값을 넣음

**2\_1) 자료구조 :**

**- thread구조체**

**- 이외 동기화를 위한 semaphore**

**2\_2) 알고리즘**

**read/write함수**

Synchronization problem을 해결해야한다. Write할 때 Read하거나 read할 때 Write하는 상황, Write할 때 Write하는 상황을 방지해야한다. 이를 해결하기 위해, 강의자료에 나와있는 Readers-writers problem의 코드를 참조하여 작성하였다.

두 개의 semaphore mutex, wrt와 readcount 변수를 사용하여 Read과 Write의 동기화를 추구하였다. deny write to executable함수를 적절히 활용해야하는데 , 이미 file이 사용되고 있을 때 denying write to executable 함수를 호출할 환경을 체크하였다.

**[thread\_ wait exit함수]**

▶semaphore\_wait라는 변수를 이용해서 동기화를 구현하였다.

wait에서는

parent-child관계가 없을 때 부모가 wait하는건 아닌지 (contain이라는 지역변수로)

이미 기다리고 있는 parent-child관계는 아닌지 ( cur -> iswait라는 구조체변수로)

이미child가 죽었고 부모가 확인한건 아닌지 (cur->exit\_accepted 라는 구조체변수로)

등에 대한 예외처리로 불필요한 wait를 방지 하였다.

exit에서는 출력양식에 맞게 출력해주고 semaphore\_wait를 up해준다

▶process\_Exit (혹은 thread\_Exit) => Orphan 문제해결

비정상적인 wait와 exit으로 zombie / orphan의 문제가 발생하는 상황이 있음을 발견하였다. 이는 time\_slice가 cpu burst보다 짧아서 생길 수 있는 문제였다.

즉 , parent가 thread를 만들고 context switching이 일어나서

wait를 부르기 전에 child가 수행을 하고 이미 exit을 한 상태가 될 수 가 있다.

“sema\_orphan 이라는 변수는 내가 죽기 전에, 부모의 sema\_orphan을 down하여 아직 자식이 있으니 부모가 죽지 말고기다리라는 것을 의미한다. 나의 자식이 죽을 때 나의 sema\_orphan을 up해주어 비로소 내가 죽을 수 있을 환경을 제공해준다“

여기서 조심해야할 점은 main thread와 idle thread를 제외하여 유의미한 부모-자식간의 관계에서만 이러한 작업이 진행되어야한다. (main과 idle에서는 해당사항이 없다)

**[파일 시스템]**

**◉ syscall\_handler에서 사용할 sema\_phore 초기화**

**◉ open /close**

**-open]**

▶ file.h에 define된 FILE\_OPEN\_LMT라는 변수를 활용할 것이다

이는 #define FILE\_OPEN\_LMT 35라고 되어있고, multi-oom 에러와 관련하여 열 수 있는 파일 개수를 제한시킨다. 따라서 메모리 범람을 막는 역할을 한다.

▶동적할당(malloc)을 통해 공간을 확보하고

filesys\_open함수를 이용해서 파일을 연다.

▶여기서 file\_deny\_Write라는 함수를 사용하였다. file\_deny\_write라는 함수는 이미 FILE이 closed되었거나 이미 열려진 파일에 write를 방지하기 위해서 사용이 되었다.

▶fd를 부여하였다. sys\_fd라는 전역변수를 이용하여 작성하였는데 새로 open할 때마다 sys\_fd를 하나씩 증가 시켰고, 이 값을 만든 file 구조체의 fd에 저장하였다. 이때 lock\_aquire lock\_release를 사용하여 atomic process로 만들어서 ++가 assemble어로 변환되어 수행할 때 가져올지도 모르는 문제를 방지하였다.

▶ 이렇게 만들어진 file을 현재 thread의 file\_list에 추가하였다.

**-close]**

parameter로 넘어온 fd를 close하는 함수이다.

▶현재 thread의 file \_list에서 해당하는 file\_elem을 찾고 이를 file\_close해준다음 list\_remove이용해서 thread의 file\_list에서도 삭제해준다.

▶open시 malloc해준 영역을 free시켜준다

**◉create**

filesys\_create 함수를 이용하여 구현

**◉remove**

filesys\_remove 함수를 이용하여 구현

**◉filesize**

filesys\_filesize 함수를 이용하여 구현

**◉tell**

thread\_current()를 통해 현재 쓰레드를 찾아낸 후 그 쓰레드의 file\_list에서 fd와 같은 file\_descriptor를 가진 file을 찾아낸다. 이 file에 대하여 file\_seek함수를 호출하여 fd 파일에서 다음에 읽거나 쓰일 위치를 position의 위치로 변환하여 준다.

**◉seek**

thread\_current()를 통해 현재 쓰레드를 찾아낸 후 그 쓰레드의 file\_list에서 fd와 같은 file\_descriptor를 가진 file을 찾아낸다. 이 file에 대하여 file\_tell함수를 호출하여 fd 파일에서 다음에 읽거나 쓰일 위치를 반환하도록 한다.

**\*\*exception 해결**

bad-read문제   
load에 대한 문제라는 것을 파악했고, 이는 make.config file 수정을 통해 해결

**3. 시험 및 평가 내용:**

- 평가 방법에 대한 설명을 기술하라.(수행 sequence 등)

pass tests/filesys/base/syn-write

pass tests/userprog/args-none

pass tests/userprog/args-single

pass tests/userprog/args-multiple

pass tests/userprog/args-many

pass tests/userprog/args-dbl-space

pass tests/userprog/sc-bad-sp

pass tests/userprog/sc-bad-arg

pass tests/userprog/sc-boundary

pass tests/userprog/sc-boundary-2

pass tests/userprog/halt

pass tests/userprog/exit

pass tests/userprog/create-normal

pass tests/userprog/create-empty

pass tests/userprog/create-null

pass tests/userprog/create-bad-ptr

pass tests/userprog/create-long

pass tests/userprog/create-exists

pass tests/userprog/create-bound

pass tests/userprog/open-normal

pass tests/userprog/open-missing

pass tests/userprog/open-boundary

pass tests/userprog/open-empty

pass tests/userprog/open-null

pass tests/userprog/open-bad-ptr

pass tests/userprog/open-twice

pass tests/userprog/close-normal

pass tests/userprog/close-twice

pass tests/userprog/close-stdin

pass tests/userprog/close-stdout

pass tests/userprog/close-bad-fd

pass tests/userprog/read-normal

pass tests/userprog/read-bad-ptr

pass tests/userprog/read-boundary

pass tests/userprog/read-zero

pass tests/userprog/read-stdout

pass tests/userprog/read-bad-fd

pass tests/userprog/write-normal

pass tests/userprog/write-bad-ptr

pass tests/userprog/write-boundary

pass tests/userprog/write-zero

pass tests/userprog/write-stdin

pass tests/userprog/write-bad-fd

pass tests/userprog/exec-once

pass tests/userprog/exec-arg

pass tests/userprog/exec-multiple

pass tests/userprog/exec-missing

pass tests/userprog/exec-bad-ptr

pass tests/userprog/wait-simple

pass tests/userprog/wait-twice

pass tests/userprog/wait-killed

pass tests/userprog/wait-bad-pid

pass tests/userprog/multi-recurse

pass tests/userprog/multi-child-fd

pass tests/userprog/rox-simple

pass tests/userprog/rox-child

pass tests/userprog/rox-multichild

pass tests/userprog/bad-read

pass tests/userprog/bad-write

pass tests/userprog/bad-read2

pass tests/userprog/bad-write2

pass tests/userprog/bad-jump

pass tests/userprog/bad-jump2

pass tests/userprog/no-vm/multi-oom

pass tests/filesys/base/lg-create

pass tests/filesys/base/lg-full

pass tests/filesys/base/lg-random

pass tests/filesys/base/lg-seq-block

pass tests/filesys/base/lg-seq-random

pass tests/filesys/base/sm-create

pass tests/filesys/base/sm-full

pass tests/filesys/base/sm-random

pass tests/filesys/base/sm-seq-block

pass tests/filesys/base/sm-seq-random

pass tests/filesys/base/syn-read

pass tests/filesys/base/syn-remove

pass tests/filesys/base/syn-write

- 자신들의 결과물이 갖는 **보건 및 안정**, **생산성과 내구성**에 대하여 반드시 기술할 것.

**<MAKE CHECK 수행화면>**

**<MAKE　GRADE 수행화면>**

**V. 기타**

- 기타 관련 내용을 기술할 것.

**1.** **연구 조원 기여도**: 기여도를 백분율로 표현할 것.

권나영 50%, 박상욱 50%

**2.** 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

권나영 :