**Pintos Project 2: User Program (2)**

담당 교수 / 분반 :서강대학교 김영재

이름 / 학번 : 진상우 / 20151378

개발 기간 : 2020.11.10 – 2020.11.17

1. **개발 목표**

* Pintos에서 User Program의 활용도 및 이해도를 높이고, system call을 구현한 뒤에 syscall\_handler에 이를 추가하고, read와 write가 파일 관련 기능을 할 수 있게 추가적으로 만들고 같은 파일을 동시에 사용하지 않기 위하여 synchronization을 구현한다. 또한 wait와 exit에서 동시에 사용하기 위해 synchronization은 이전의 busy-waiting에서 semaphore를 이용한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor

2. (이번 프로젝트에서 구현해야 하는) System Calls

3. Synchronization in Filesystem

Src/thread의 thread.h, thread.c

Src/userprog syscall.h syscall.c process.c

Src/filesys filesys.c filesys.h file.c file.h

* SYSTEM CALLS

Create, remove, open, close, filesize, seek, tell을 구현하고, 이전의 read, write를 고쳐 syscall.c에 구현한다.

* Synchronization
* Wait과 exit에서 semaphore를 사용한 synchronization이 일어나게 이전의 busy waiting을 수정하여 process.c의 process\_wait,process\_exit을 수정하고, semaphore에 대한 자료구조를 나타낸다. 그리고 load와 exit에서 load한 file을 종료하기 전 다른 프로세스에서 수정 못하게 file\_deny\_write, file\_allow\_write을 통해 막고, process.c에서 한다.
* File descriptor
* 프로세스가 메모리에 로드되어 실행되어도 실행중 프로그램의 file이 지워지지 않도록 한다. File에 write을 허용하거나 허용하지 않으며 구현한다.
  1. **개발 내용**
* 아래 항목의 내용만 서술
  + 1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

실행중에 프로그램의 executable file이 지워지지 않도록 구현하였고, file\_deny\_write, file\_allow\_write함수가 잘 사용되게 하며, file이 load될 때, file\_deny\_write함수를 호출하여 file에 쓰기를 못하게 설정하고, thread가 종료될 때, file에 쓰기를 허용하여 executable file이 지워지지 않게 하였다.

2. System Calls: 구현할 각 system call에 대해 간략히 서술 (하나의 system call 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)

Create: sys\_create를 이용해 filesys\_create, system call을 나타낸다

(file이름, file size)

Remove: sys\_remove를 이용해 filesys\_remove, system call을 나타낸다

Parameter를 이름으로 가지는 file을 제거한다.

Open: sys\_open를 이용해 filesys\_open, system call을 나타낸다

Parameter를 이름으로 가지는 file을 찾아 open, 파일이 open되면, file\_list에 추가하고, find\_executable\_file함수를 이용해 이름이 같은 파일이 존재하면 file\_deny\_write를 호출한다.

Close: sys\_close를 이용해 filesys\_close, system call을 나타낸다

Parameter를 이름으로 가지는 file을 찾아 close

Filesize : sys\_filesize를 이용해 file\_length, system call을 나타낸다

Parameter에 해당하는 file의 size를 return

Seek: sys\_seek를 이용해 system call, file\_list가 null이 아니면 file\_seek를 call

처음의 parameter의 file description의 file을 찾아 그 위치를 두번째 parameter의 position으로 옮긴다

Tell : sys\_tell을 이용해 system call, file\_tell을 call한다.

Parameter에 해당하는 file을 찾아 그 위치를 return

Read : sys\_read를 통해 system call을 하고, fd가 0일 때, input\_getc, fd>2일 때 file\_read.

Standard input을 읽는 동작에 file을 읽는 것을 추가한다.

Write: sys\_write를 통해 system call, fd가 1일 때, putbuf를 호출, size를 return, else if file\_write call

Standard output 출력하는 동적에 file에 write하는 것 추가한다.

3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)

Synchronization은 semaphore를 이용하여 critical section에 코드가 접근 하기 직전 압뒤로 semaphore value를 각각 함수를 이용해 증가 혹은 감소 시키며 critical section에 접근 하는 thread수를 제한 해주며, 0,1만을 사용할 수 있고, 1이상의 수를 가지기도 한다.

Single thread에서는 값을 1로 초기화하여 하나의 thread가 critical section에 도착 할 때 그 값을 1 감소시킨 뒤 0이 되면 해당 thread가 종료되기 전까지 해당 ciritical section에 접근 하는 다른 thread를 block한다. 접근 하던 thread가 종료되면 다시 1 증가시켜 block 상태의 thread 중 하나를 unblock한다. 다시 unblock된 thread가 접근하며 1 감소하며 이게 반복된다. Multi-thread에서는 이를 1보다 크게 설정해 critical section에 한번에 접근하는 thread 수를 제한한다.

이렇게 전후로 lock을 얻고 풀어주며 관련 함수를 call한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2020.11.10 – 2020.11.14 : 지난 프로젝트1을 완전히 하지 못하여 그를 완성하고, 각 system call을 분석하고, 매뉴얼도 보며 어떻게 해나가야할지 계획하였다.

2020.11.14 – 2020.11.17 : 본격적으로 system call을 구현해 나가고, file 관련 파일도 구현하고, wait와 exit함수를 synchronization을 이용하였고, lock과 semaphore를 사용하였다.

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명

Create: file 이름의 새로운 파일을 size만큼 생성한다. File 이름이 NULL이면, exit(-1)을 호출하여 thread를 종료하여 예외처리하고, Null이 아니면 filesys\_create를 호출한다. 이 함수의 parameter는 파일이름과 초기의 크기이며, create를 성공하면 true, 실패하면 false를 나타낸다. 하지만 이 함수는 파일을 열지는 않는다.

Remove : file이라는 파일을 지우고, 지우는 것은 filesys\_remove 함수를 이용하여 수행하는데, 파일이 성공적으로 delete가 되면 true, 그렇지 않으면 false를 return한다. 파일이 열려있든 닫혀있든 상관없이 삭제하기에 파일 자체의 존재 여부는 검사할 필요가 없이 filesys\_remove를 호출하며 파일을 닫지는 않는다.

Open: 이는 file을 열어주는 작업을 수행하며, 두가지를 검사하면서 open하는데, 먼저 fd를 검사하고, 다음은 opening\_file의 존재를 확인한다. 먼저 0 or 1인 fd는 stdin.out이 차지하고, fd 0은 standard input, fd 1은 standard output이다. 그래서 open system call 구현 할 때, 만들어진 file의 수인 fd를 보고 0이면 2로 초기화를 한다 이는 console의 file descriptor를 피하기 위함이고, 두번째로는 파일이 성공적으로 open되지 못할 경우엔 -1을 return하고, 성공적으로 open될 경우엔 file\_list에 open한 file을 추가하고, file\_deny\_write함수를 호출하여 현재의 thread가 저장된 all\_list를 검색하여 이름이 같은 file이 존재하면, file\_deny\_write를 해준다. 각 프로세스는 독립적인 file descriptor set을 가지기에 하나의 file이 하나의 프로세스나 다른 여러 개의 프로세스에 의해 열리면, fd를 이전 값에서 1을 증가시키며 file\_list에 추가한다.정상적으로 수행되면 file descriptor를 return한다.

Close: fd를 parameter로 받아 thread의 file\_list를 검사해 file이 존재하면, file\_close를 호출해 file을 닫아주고, file\_list에 닫은 file이 존재한 file\_list의 element를 NULL로 한다.

Filesize: fd를 받아서 open된 file의 size를 byte단위로 return하고, 현재의 thread를 받아 thread가 가지는 file\_list에 파일이 있으면 file\_length를 호출하여 size를 받고, return하고 만약 비어있다면 0을 return한다.

Read: fd를 받아서 open된 file의 size byte만큼을 buffer에 읽고, fd가 0일 때 input\_getc, fd>1일 때 file\_read를 호출한다. Fd가 1이거나 0미만일 경우엔 error로 -1을 return 한다. 또한 phys\_base를 넘어가는지 보고 넘으면 user\_exit을 통해 나온다. 이를 모두 만족하면 file\_list를 검사해 파일이 존재하면 file\_read를 통하여 byte를 받아온다. 이때 lock을 걸어 file에 접근해 파일의 내용을 읽는 수행이 critical section으로 어떠한 프로세스가 파일을 읽고 있는데 다른 프로세스가 접근하는 것을 막는다.

Write : fd를 받아서 open된 file에 buffer로부터 size만큼 가져와 write한다. Read와 비슷하지만, fd가 1일 때, putbuf를 호출하고, fd가 2이상일 경우에, file\_list에 파일이 존재하면 file\_deny\_check를 사용한다. 이는 현재 file이 write가 deny되어있는지 check한다. 그렇다면, true, 아니면 false를 return하고, 이 함수를 쓰는 것은 parent가 open, read, write하고 exec를 할 때 child를 생성하는데 이경우 child가 parent의 file에 쓰는 것을 막아주기 위함이고 나머지는 read와 같이 lock을 걸고 file\_write를 수행한 후 lock을 풀어준다.

Tell: fd를 받아 open file fd에서 읽거나 쓰일 next byte를 return하고, 이는 next\_byte를 return 하는 함수인 file\_tell을 호출하여 값을 받아온 뒤에 이를 return한다.

Seek: 이 system call은 open file fd에서 읽거나 쓸 next byte를 position으로 바꾸는데, 이는 다음 byte를 찾아 position을 return 해주는 역할을 하며 현재의 thread의 file\_list에 file이 있을때 file\_seek의 parameter로 넘겨 호출한다.

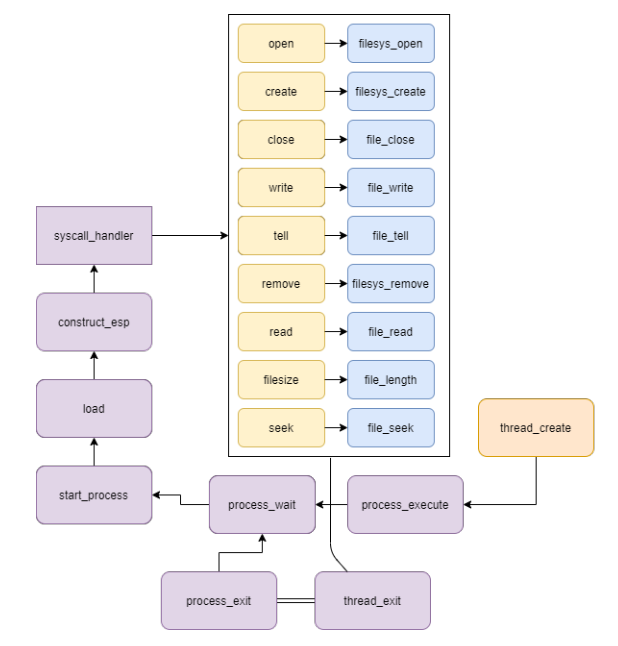
* Synchronization

각각의 thread마다 semaphore 2를 두어 각각 wait와 exit을 구성한다. Wait\_sema는 process\_wait에서 인수로 넘어온 child\_tid와 현재 thread의 child list의 tid와 같으면 child list의 wait\_sema에 대해 sema\_down을 수행한다. 그리고 child list가 exit할 때, parent의 wait list가 차있고, child tid와 parent가 종료할 child\_tid가 같을 때, child thread의 wait\_sema에 관하여 sema\_up을 한다. 그 이후 thread\_exit에서 parent thread의 exit\_sema의 wait\_list에 list가 남아있을 때, wait list의 모든 list에 관해 sema\_up을 모두 수행하여 parent도 exit을 한다. 이러한 exit\_sema는 process\_exit함수에서 종료하는 thread가 main thread가 아니고, 종료하려는 thread의 parent가 unblock 상태일때, 현재 thread의 parent의 exit\_sema에 대해 sema\_down을 한다. 또, open, read, write에 대해 함수 실행 전후로 lock을 걸고 풀어주는 행동을 취한다.

* Execution

Load 함수에서 file이 load되었을 때, file\_deny\_write를 호출하여 file에 write할 수 없게한다. 그리고 denied 배열에 file을 추가하고, thread가 종료될 때 process\_exit 에서 file에 write를 해줄 수 있게 한다. 이때 file이 deny\_write한게 여러 개 존재할 수 있기에 denied 배열에 동일한 file이 존재하는지 보고, 그렇다면 함수를 호출하지 않고, denied 배열에서만 지우고, 동일한 file이 존재하지 않는다면 함수를 호출하고, 배열에서 삭제한다.

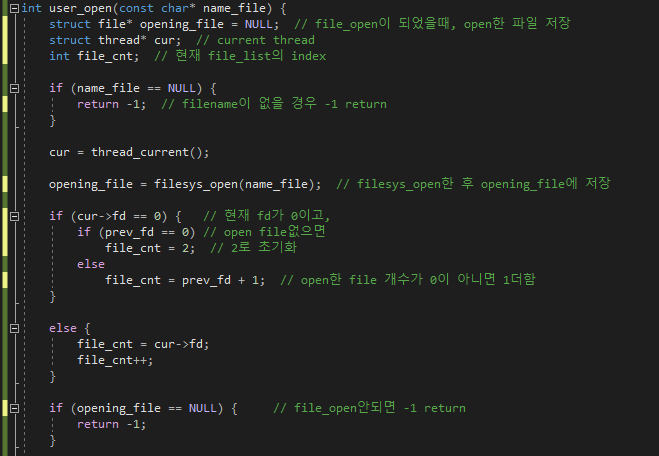
1. **연구 결과**
   1. **Flow Chart**

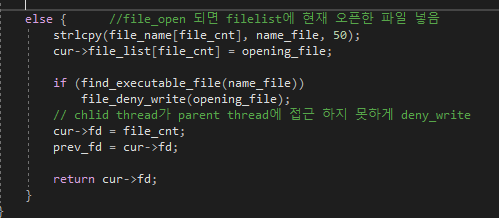
****

Synchronization은 file open, read, write에 수행, file system call에서는 create, remove, open. Open이 성공적으로 file을 열면, filedescriptor를 file에 부여, 이를 받아 수행하는건 close, read, write, seek, tell, filesize이다. 또, close가 file을 닫으면 현재 thread의 fd를 가진 file을 할당함으로 관리한다.

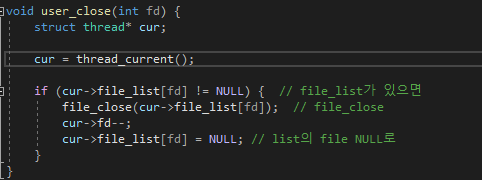
* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명
* **Userprog/syscall.c**

User open- FILE SYSTEM CALL

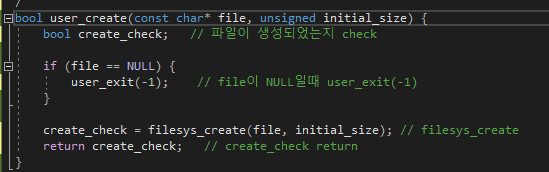




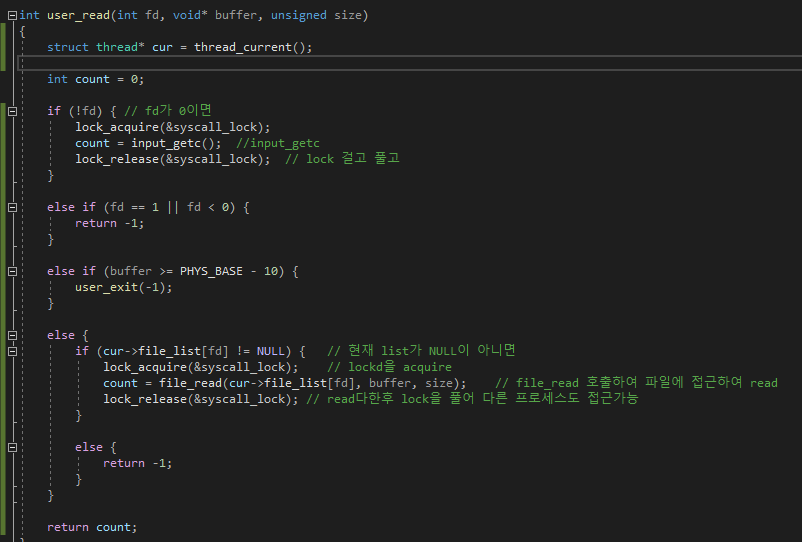
User\_close- FILE SYSTEM CALL



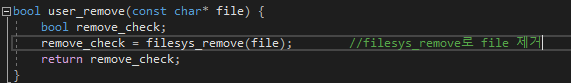
User\_create- FILE SYSTEM CALL



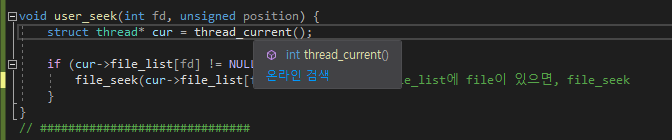
User\_read- FILE SYSTEM CALL



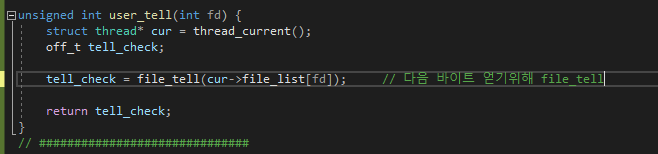
User\_remove- FILE SYSTEM CALL



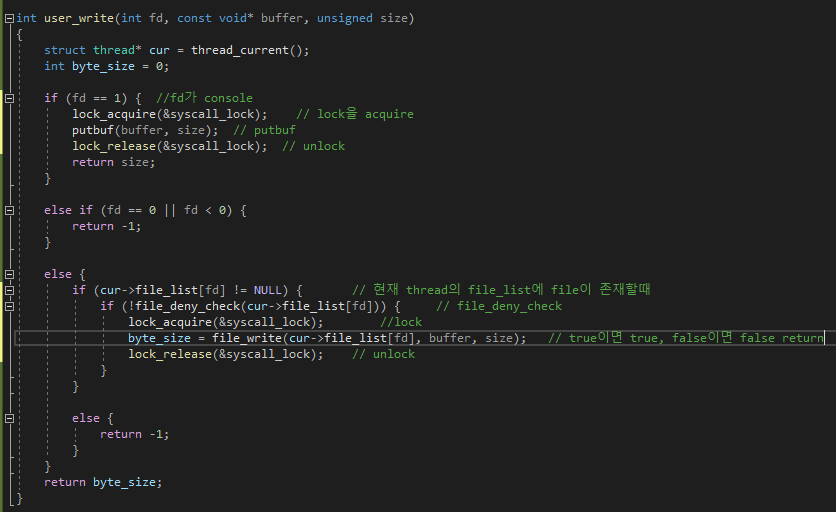
User\_seek- FILE SYSTEM CALL



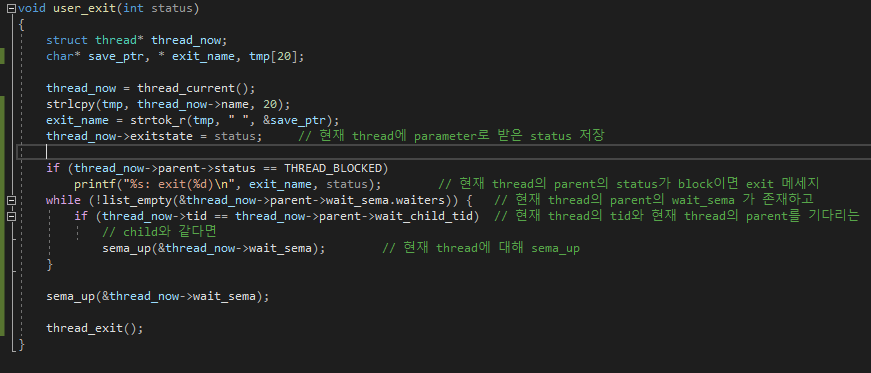
User\_tell- FILE SYSTEM CALL



User write- FILE SYSTEM CALL

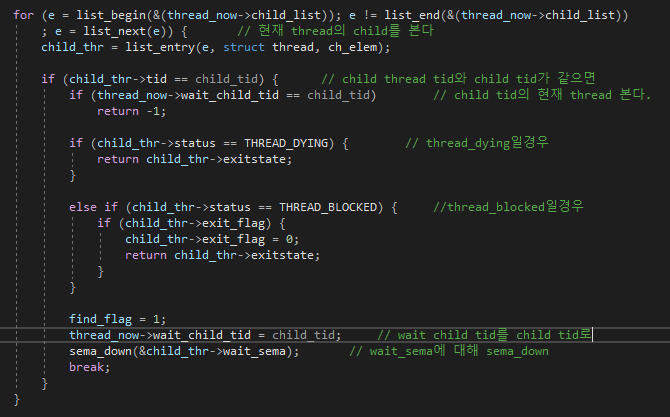


User\_exit – synchronization

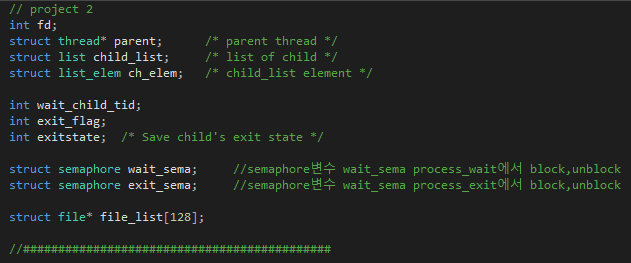


* **Userprog/process.c**

Process\_wait - synchronization

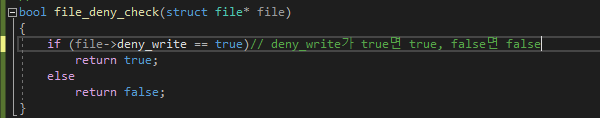


Thread/thread.h -synchronization



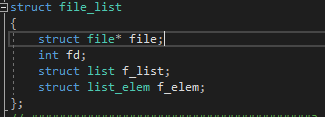
**Filesys/ file.c**

File\_deny\_check 함수 생성



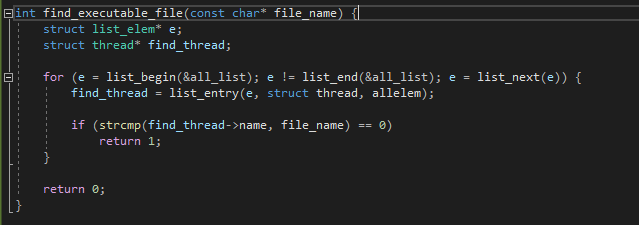
**Filesys/file.h**

File\_list에 관한 변수들 설정

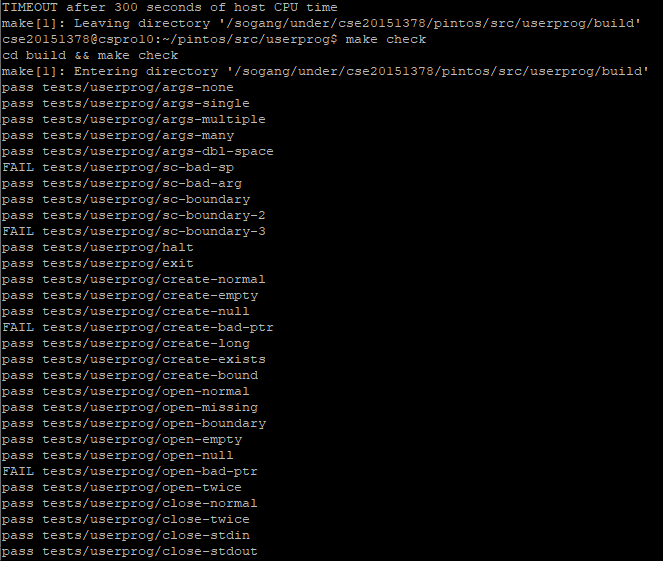
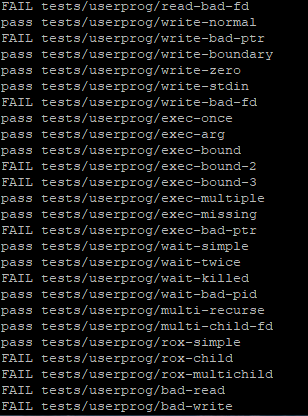


**Thread/thread.c**

Find\_executable \_file



실행가능한 파일을 찾는 함수

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 
* 
* 