**Pintos Project 1: User Program (1)**

담당 교수 : 박성용 교수님

학번 : 20181702

이름 : 홍주표

개발 기간 : 2022.09.18 ~ 2022.10.03

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

Pintos 환경에서 주어진 베이스라인 코드를 이용하여 user program을 구현한다. 실행시킬 argument를 user stack에 passing하고 system call을 수행한다. 본 프로젝트에서는 halt, exit, exec, wait, read(stdin), write(stdout)만을 구현한다. 추가적인 system call로 해당 값의 피보나치 값을 구하는 fibonacci와 4개의 숫자 중 최댓값을 구하는 max\_of\_four\_int를 새로운 system call로 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

입력받은 argument를 스페이스 단위로 parsing 한 후, 그들을 user stack에 쌓는다.

1. User Memory Access

메모리에 접근하는 과정에서 주소가 user virtual memory에 있는지 확인하고, 주소가 잘못 넘겨지게 된다면 프로그램을 종료한다.

1. System Calls

user stack에 있는 명령어를 system call handler와 그에 맞는 각각의 함수들을 통해 적절히 처리한다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing
  + 커널 내 스택에 argument를 쌓는 과정 설명

먼저 baseline에 있는 filesys\_open 이전에 입력받은 argument를 space 단위로 parsing한다. 앞에서부터 차례대로 argv[0], argv[1], … 에 저장하고 그 개수인 argc도 저장한다.

그리고 setup\_stack 함수가 호출된 후 넘겨야 할 인자들을 스택 형식으로 쌓는다. 먼저 \*esp = PHYS\_BASE로 초기화한다. 그 후 argv[i], word align, NULL, argv[i] 각각의 주소값, argv의 주소값, argc, return address를 순서대로 스택에 쌓는다. 이 때 각각의 size에 맞게 esp 값을 적절히 감소시키며 스택을 형성한다. 자세한 사항은 뒤에 나올 항목에서 서술하겠다.

* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명
  + Invalid memory access를 어떻게 막을 것인지 설명

user mode에서 kernel mode의 memory 영역에 참조하려고 하는 것을 pintos 상에서의 invalid memory access라고 한다.

invalid memory access를 막기 위해서 pintos/src/userprog 내의 syscall.c에서 각 system call number에 맞는 system call 함수를 호출하기 전에 is\_user\_vaddr 함수를 사용하여 처리한다. 그 값이 False라면 system call 함수를 호출하지 않고 exit(-1)을 하여 프로그램을 종료한다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명
  + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

시스템 콜은 user mode 영역에서 kernel mode의 유용한 API들을 사용 가능하게 하기 위해서 필요하다.

이번 프로젝트에서는 halt, exit, exec, wait, read, write, fibo, max에 대한 시스템 콜을 구현한다. 먼저 sys\_num이라는 변수를 선언하여 system call number를 할당한다. 그리고나서 switch문을 이용하여 각각에 맞는 기능을 구현한다. system call 함수를 구현하기 전에 앞서 설명한 invalid memory access를 막기 위한 코드도 삽입한다.

* + 1. SYS\_HALT

pintos를 종료하는 시스템 콜이다. halt 함수를 이용하여 구현하며 halt 함수에서 shutdown\_power\_off() 함수를 사용한다.

* + 1. SYS\_EXIT

현재 user program을 종료하고 커널에 현재 상태(status)를 반환하는 시스템 콜이다. exit 함수를 이용하여 구현하며 exit 함수에서 thread\_exit() 함수를 사용한다.

* + 1. SYS\_EXEC

실행을 하기 위해 child thread를 만드는 시스템 콜이다. exec 함수를 이용하여 구현하며 exec 함수에서 process\_execute() 함수를 사용한다.

* + 1. SYS\_WAIT

parent thread가 child thread보다 먼저 끝나지 않기 하기 위한 시스템 콜이다. wait 함수를 이용하여 구현하며 wait 함수에서 process\_wait() 함수를 사용한다.

* + 1. SYS\_READ

열려있는 파일을 읽는 시스템 콜이다. 이 프로젝트에서는 STDIN에 대해서만 다룬다. read 함수를 이용하여 구현하며 read 함수에서 input\_getc() 함수를 사용한다.

* + 1. SYS\_WRITE

파일에 데이터를 입력하는 시스템 콜이다. 이 프로젝트에서는 STDOUT에 대해서만 다룬다. write 함수를 이용하여 구현하며 write 함수에서 putbuf() 함수를 사용한다.

* + 1. SYS\_FIBO

입력한 값에 대한 피보나치 수를 구하는 시스템 콜이다. fibonacci 함수를 이용하여 구현하며 fibonacci 함수에서 재귀가 아닌 for 문을 사용하여 값을 구한다.

* + 1. SYS\_MAX

max\_of\_four\_int(네 정수 중 최댓값 구하기)를 위한 시스템 콜이다. max\_of\_four\_int 함수를 이용하여 구현한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**

22.09.18 ~ 22.09.21: Argument Parsing 및 Passing, Stack 구현

22.09.22 ~ 22.09.28: Invalid Memory Access 및 System Call 구현

22.09.29 ~ 22.10.03: 추가 구현 및 보고서 작성

* 1. **개발 방법**
* **II. B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**
  + 1. pintos/src/userprog/process.c 중 load 함수에서 argument passing을 구현한다.

baseline에 있는 filesys\_open 이전에 입력받은 argument를 space를 기준으로 parsing한다. 앞에서부터 차례대로 argv[0], argv[1], … 에 저장하고 그 개수인 argc도 저장한다. 이 때 파라미터로 넘겨받은 file\_name을 strlcpy 함수를 이용하여 file\_name\_copy에 복사한 뒤, 이를 strtok\_r 함수를 이용하여 argv와 argc 값을 저장한다. 명세서의 예시로는 ‘echo x’가 argument로 들어왔다. 이 경우에는 argv[0]에 ‘echo’를, argv[1]에 ‘x’를, argc에 2를 저장한다.

추가로 기존에 있던 file = filesys\_open(fie\_name)을 file = filesys\_open(argv[0])으로 변경한다. ‘echo x’가 들어왔을 때 명령어로 ‘echo’를 입력해줘야지, ‘echo x’를 입력해주는 것이 아니기 때문이다.

그리고 setup\_stack 함수가 호출된 후 넘겨야 할 인자들을 스택 형식으로 쌓는다. 스택을 쌓기 전에 우선 \*esp를 PHYS\_BASE로 초기화한다. 첫 번째로 space 단위로 나눈 argv들을 역순으로 스택에 넣는다. 스택 포인터의 위치(esp로 후술)는 ‘strlen(argv[i]) + 1’씩을 감소하며 지정한다. 그 후 memcpy를 이용하여 해당 esp에 argv들을 저장한다. 또한 이후에 저장할 word align과 argv들의 주소를 위해 각각 total\_len 변수 처리 및 argv[i]를 esp로 대치했다. 두 번째는 word align이다. 4칸을 맞춰주기 위함이다. 앞에서 argv[i]를 저장할 때 사용한 total\_len 변수를 이용하여 word align을 처리한다. 세 번째로 NULL 값을 저장한다. 네 번째로 argv[i] 각각의 주소값을 저장한다. 첫 번째 과정 마지막에 argv[i] 값들을 그들의 주소값으로 바꿨기 때문에 바로 그들을 사용한다. ‘\*\*(uintptr\_t \*\*)esp = (uintptr\_t)argv[i]’ 와 같은 코드를 사용한다. 다섯 번째로 argv의 주소를 저장한다. 이는 바로 위 칸(esp + 4)의 내용을 저장하면 된다. 여섯 번째로 argc 값을 저장한다. 앞서 argument parsing 과정에서 argc를 구했기 때문에 이 값을 그대로 사용한다. 마지막으로 return address를 저장한다. 이로써 user stack 구현을 마친다. 세 번째 과정부터는 esp 값을 4씩 감소시키며 저장한다. 또한 각 스택에 들어오는 값의 type에 맞게 esp의 타입을 캐스팅하여 오류나 warning을 없앤다.

* + 1. pintos/src/userprog/syscall.c에서 system call을 구현한다.

먼저 syscall\_handler 함수에서 f->esp를 이용하여 명령어에 해당하는 system call number 값을 sys\_num 변수에 저장한다. 이를 switch 문을 이용하여 각 system call에 맞게 나눈다. switch 문은 SYS\_HALT, SYS\_EXIT, SYS\_EXEC, SYS\_WAIT, SYS\_READ, SYS\_WRITE, SYS\_FIBO, SYS\_MAX로 나누어져 있다. 각 case마다 is\_user\_vaddr 함수를 사용하여 invalid memory access를 처리한다. 알맞게 들어온 주소값이라면 각 case에 맞는 함수를 호출하여 system call을 수행한다.

* + 1. pintos/src/userprog/exception.c 중 page\_fault 함수에 user memory access에 대한 내용을 추가한다.

if(!user || is\_kernel\_vaddr(fault\_addr)) exit(-1); 코드를 추가하여 메모리 접근에 대한 에러가 발생할 때 프로그램을 종료하도록 한다.

* + 1. pintos/src/threads/thread.h 중 thread 구조체에 변수를 추가한다.

이는 synchronization을 해주기 위함이다.

pintos/src/threads/thread.c 중 init\_thread 함수에 구조체를 생성한다.

이는 child 프로세스가 끝날 때까지 parent 프로세스가 기다려주기 위함이다. 구조체는 #ifdef USERPROG ~ #endif를 사용하여 생성한다.

또한 pintos/src/userprog/process.c 중 process\_wait 함수와 process\_exit 함수에 세마포어(sema\_up, sema\_down)를 이용하여 필요한 내용을 추가한다.

* + 1. 추가 구현(fibonacci, max\_of\_four\_int)을 처리한다.

pintos/src/lib/syscall-nr.h에 두 추가 구현을 위한 system call number를 추가한다. 각각 이름은 SYS\_FIBO, SYS\_MAX로 한다.

pintos/src/lib/user/syscall.c에 인자 4개짜리 함수인 max\_of\_four\_int를 위한 syscall4 함수를 define한다. 이는 기존에 있는 syscall0 ~ syscall3을 참고한다. 또한 fibonacci 함수와 max\_of\_four\_int 함수를 추가하고 각각 syscall1 함수, syscall4 함수를 사용하여 값을 반환한다.

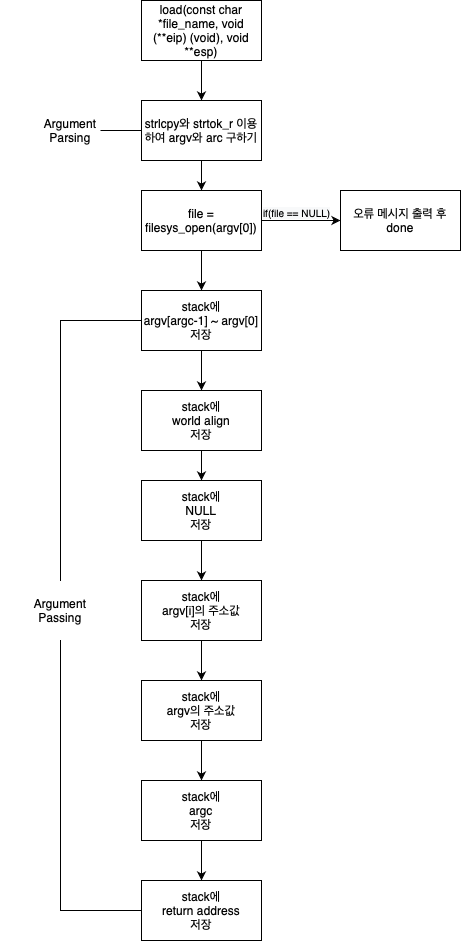
pintos/src/userprog/syscall.c에는 fibonacci 함수와 max\_of\_four\_int의 기능을 구현하는 함수를 추가한다. 피보나치는 for문을 이용한 방식으로 구현한다.

pintos/src/examples에 additional.c 파일을 추가한다. 여기에는 추가 구현 사항을 명세서에 나온 대로 출력하도록 한다. 이 내용은 기존에 존재하는 다른 파일들의 형식을 참고한다. 또한 같은 경로 내에 있는 Makefile도 additional.c를 처리할 수 있게 수정한다.

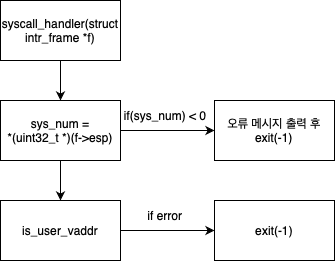
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

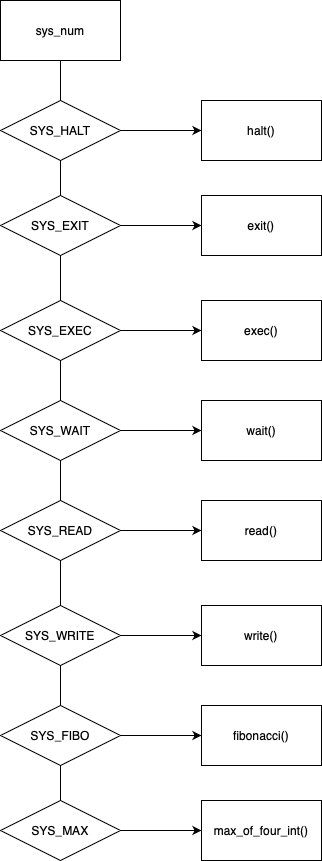
1. Argument Passing



1. User Memory Access



1. System Calls



* 1. **제작 내용**
* **II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.**
* **구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명.**
* **개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.**

1. Argument Passing
   * 1. pintos/src/userprog/process.c 중 load 함수

텍스트이(가) 표시된 사진

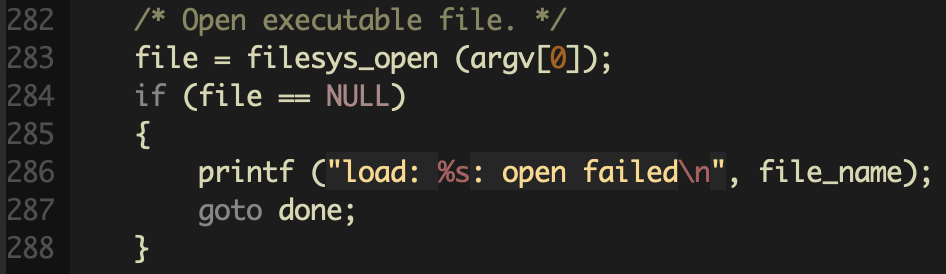
자동 생성된 설명

우선 argument parsing을 하기 위한 변수를 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후 strlcpy 함수를 이용하여 파라미터로 넘겨 받은 file\_name을 file\_name\_copy에 복사하고 strtok\_r 함수를 이용하여 argc와 argv를 저장한다.



filesys\_open의 인자로는 argument의 첫 token을 넘긴다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음으로는 스택을 쌓기 위한 변수를 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

setup\_stack 함수가 호출된 후 스택을 쌓기 시작한다.

스택에 내용을 넣기 전에 \*esp를 PHYS\_BASE로 초기화한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 순서에 맞게 스택을 쌓는다.

1. User Memory Access
2. pintos/src/userprog/syscall.c 중 process\_execute 함수

텍스트이(가) 표시된 사진

자동 생성된 설명

filesys\_open 함수와 thread\_create 함수의 인자로 argument의 첫 토큰으로 받는다. 그리고 filesys\_open을 실패하면 return -1로 프로그램을 종료한다.

1. pintos/src/userprog/syscall.c 중 syscall\_handler 함수

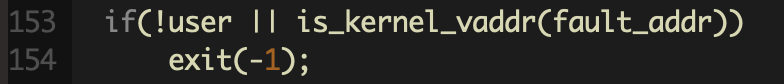
텍스트이(가) 표시된 사진

자동 생성된 설명

pintos를 바로 종료하는 system call인 SYS\_HALT를 제외한 나머지 system call에서 각각의 system call 함수를 실행하기 전에 is\_user\_vaddr 함수를 이용하여 user memory 공간인지를 확인 후 오류가 있다면 exit(-1)로 프로그램을 종료한다.

사진은 여러 system call 중 하나인 SYS\_EXIT이다.

1. pintos/src/userprog/exception.c 중 page\_fault 함수



page\_fault 내에서 invalid memory access에 대한 처리를 추가한다. 만약 user가 0이거나 fault\_addr가 kernel mode에 해당한다면 exit(-1)로 프로그램을 종료한다.

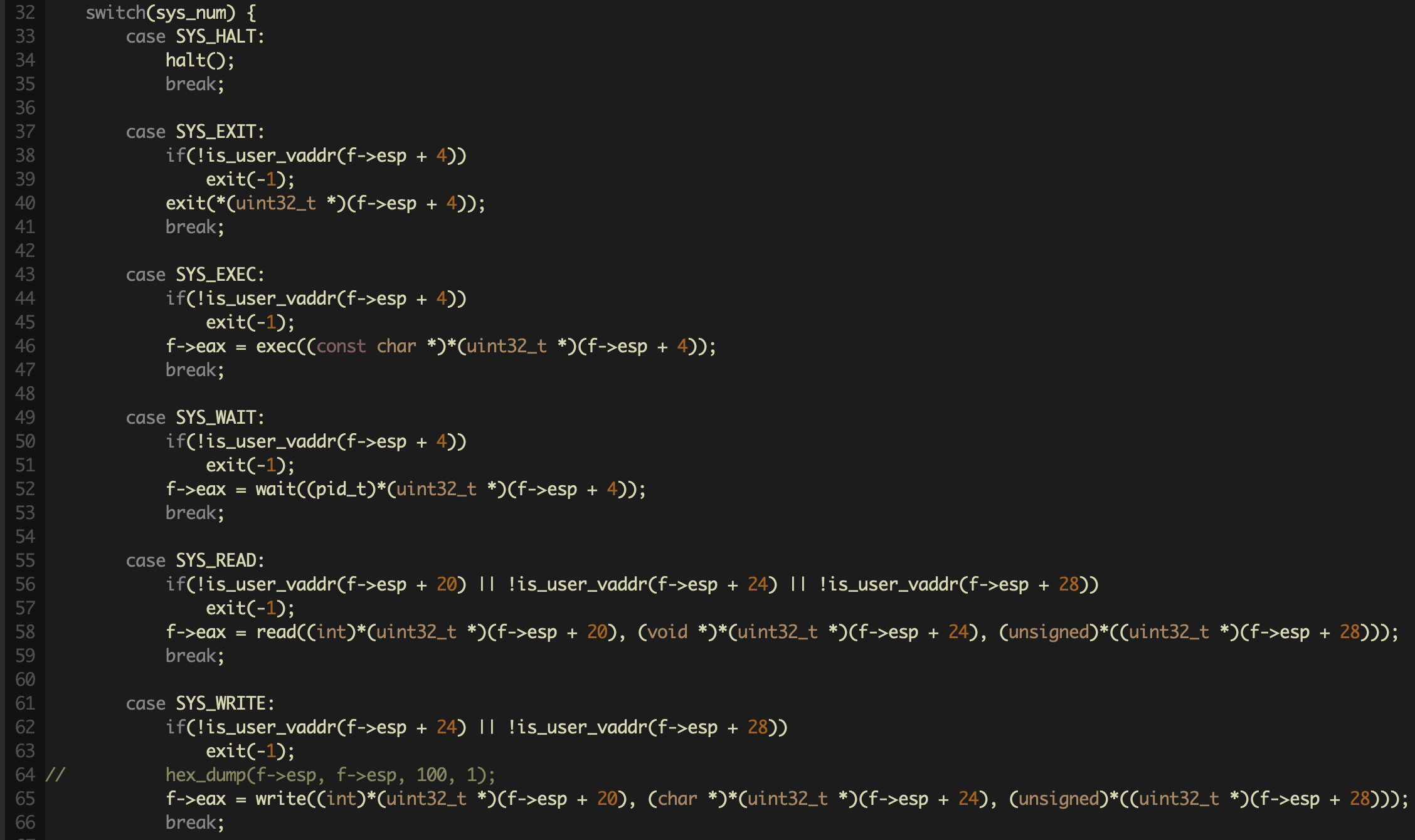
1. System Calls
2. pintos/src/userprog/syscall.c 중 syscall\_handler 함수

**텍스트이(가) 표시된 사진

자동 생성된 설명**

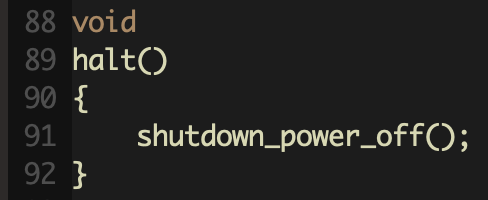
sys\_num이라는 변수를 선언하여 system call number 값을 저장한다.

이 변수 값이 음수라면 옳지 않은 값을 받은 것이기 때문에 오류메시지를 출력하고 프로그램을 종료한다.



switch 문을 이용하여 sys\_num의 값을 분류한다.

앞서 말한 user memory access에 대한 처리를 먼저 한 후, 오류가 없다면 각각에 맞는 함수를 실행한다.



SYS\_HALT에서 halt 함수를 호출한다.

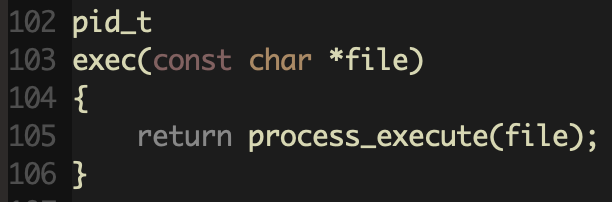
halt 함수는 shutdown\_power\_off 함수를 호출하여 처리한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

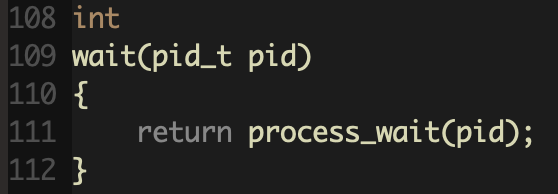
SYS\_EXIT에서 exit 함수를 호출한다.

exit 함수는 thread\_name과 status를 출력한 후, 현재 thread의 status를 변경해주고 thread\_exit 함수를 호출한다.



SYS\_EXEC에서 exec 함수를 호출한다.

exec 함수는 process\_execute 함수를 반환한다.



SYS\_WAIT에서 wait 함수를 호출한다.

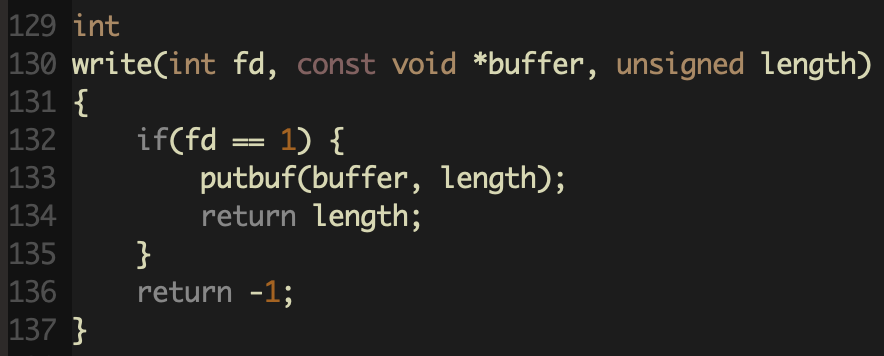
wait 함수는 process\_wait 함수를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

SYS\_READ에서 read 함수를 호출한다.

read 함수는 fd가 0일 때 input\_getc 함수를 이용하여 STDIN에서 읽어오는 기능을 한다.



SYS\_WRITE에서 write 함수를 호출한다.

write 함수는 fd가 1일 때 putbuf 함수를 이용하여 STDOUT에 출력하는 기능을한다.

1. pintos/src/userprog/syscall.h

텍스트이(가) 표시된 사진

자동 생성된 설명

각 system call 함수들의 prototype을 추가한다.

1. pintos/src/threads/thread.h

텍스트이(가) 표시된 사진

자동 생성된 설명

기존 thread 구조체에 변수를 추가한다. child\_lock과 mem\_lock은 각각 child의 상태와 memory를 lock하기 위한 구조체, child는 child thread를 lock하기 위한 구조체이다. 그리고 child\_elem과 exit\_status를 추가한다.

1. pintos/src/threads/thread.c 중 init\_thread 함수

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같은 코드를 추가한다.

1. pintos/src/userprog/process.c 중 process\_wait 함수와 process\_exit 함수

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 process\_wait 함수에서 child\_tid가 끝날 때까지 기다리고 그 후 thread의 exit\_status를 반환한다. 그리고 while 문에서 child\_lock을 세마포어 down하여 child를 기다린다. 그리고 mem\_lock을 세마포어 up하여 메모리를 없애기 전까지 남겨둔다.

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_exit 함수에서는 process\_wait와는 반대로 child\_lock을 sema\_up, mem\_lock을 sema\_down 한다.

* **이번 프로젝트에서 개발한 시스템 콜을 구현 관점에서 상세히 서술**

1. Additional System calls

* **새로운 시스템 콜(fibonacci, max\_of\_four\_int)을 구현하기 위해 수정하거나 작성한 코드에 대해 서술**
  1. pintos/src/lib/syscall-nr.h



추가 구현할 fibonacci 함수와 max\_of\_four\_int 함수를 위한 system call number를 추가한다.

* 1. pintos/src/lib/user/syscall.c

텍스트이(가) 표시된 사진

자동 생성된 설명

인자를 4개 넘겨야 하는 max\_of\_four\_int 함수를 위해 syscall4를 새로 define한다. 이는 앞서 존재하는 syscall0~3의 내용을 참고한다.

텍스트이(가) 표시된 사진

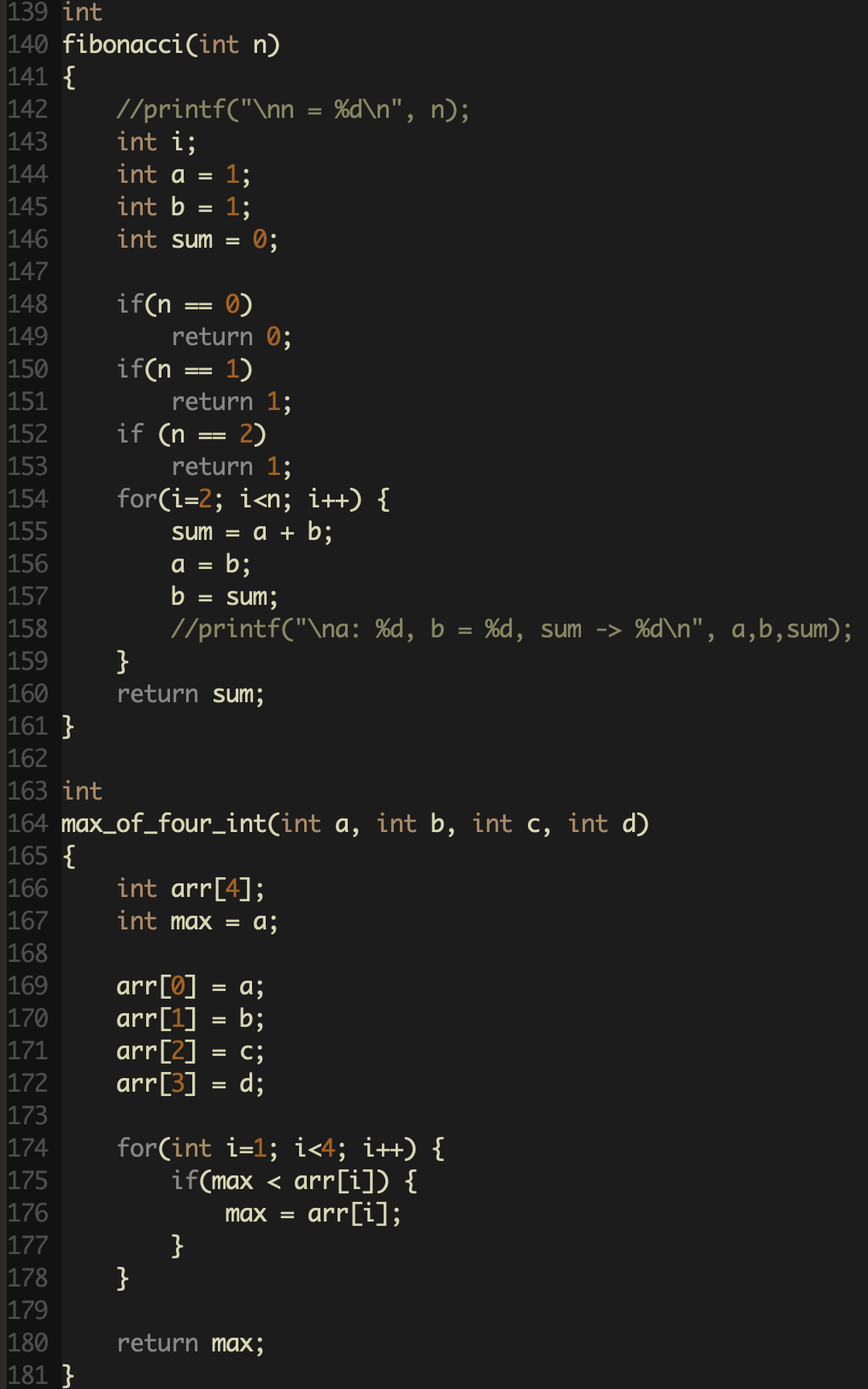
자동 생성된 설명

그리고 fibonacci 함수와 max\_of\_four\_int 함수를 추가한다. 각각 인자가 1개, 4개이기 때문에 syscall1, syscall4를 이용하여 인자를 넘겨준다.

* 1. pintos/src/lib/user/syscall.h

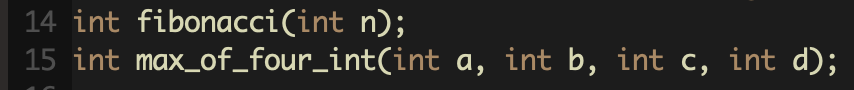
추가로 구현할 fibonacci 함수와 max\_of\_four\_int 함수의 prototype을 선언한다.

* 1. pintos/src/userprog/syscall.c



fibonacci 함수와 max\_of\_four\_int 함수의 기능을 구현하기 위한 코드를 추가한다.

* 1. pintos/src/userprog/syscall.h



두 함수의 prototype을 선언한다.

* 1. pintos/src/examples/Makefile

텍스트이(가) 표시된 사진

자동 생성된 설명

추가 구현을 실행하는데 필요한 additional.c를 실행가능하게 하기 위해 additional 관련된 코드를 추가한다.

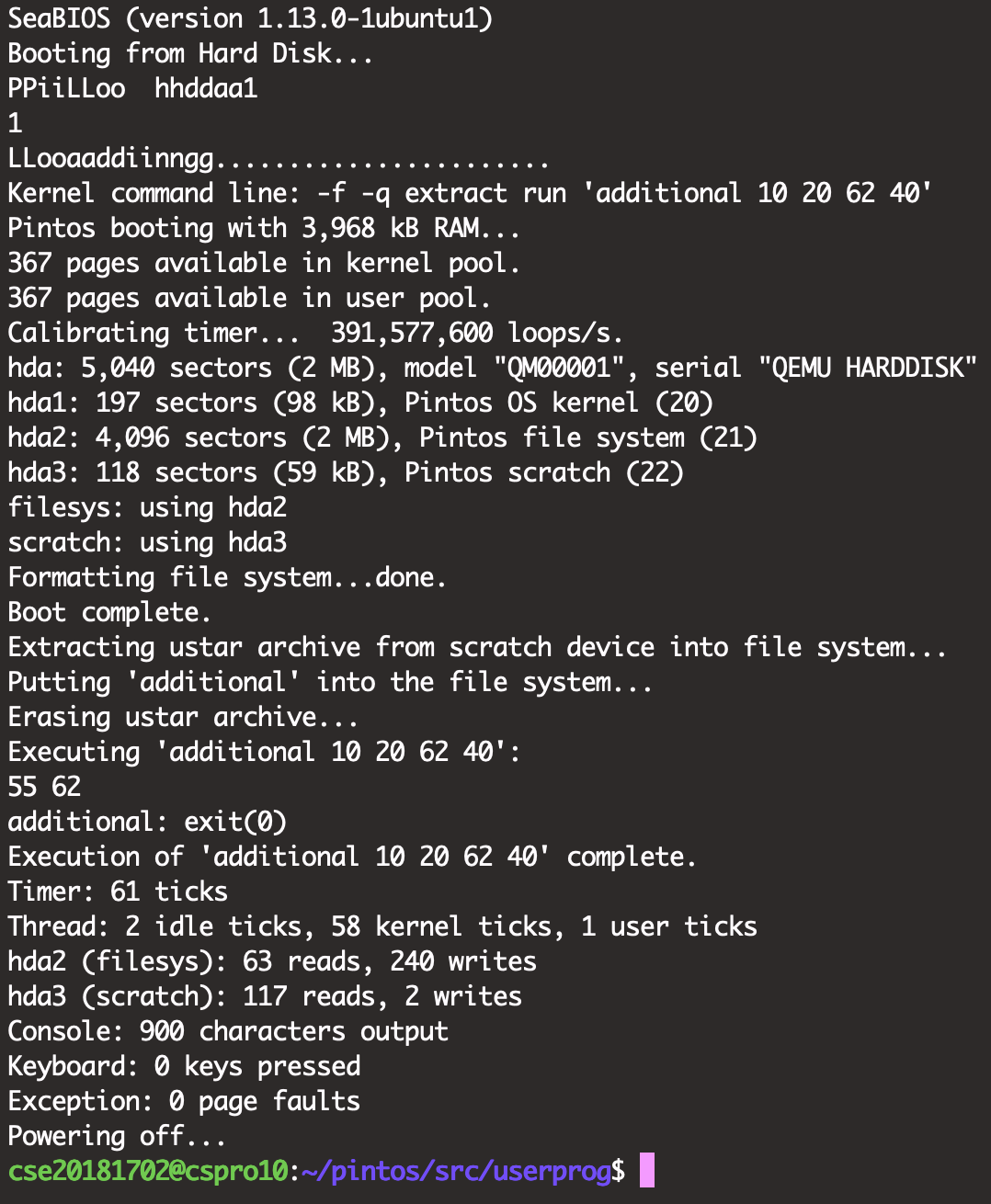
* 1. pintos/src/examples/additional.c

텍스트이(가) 표시된 사진

자동 생성된 설명

출력 형식에 맞춰 추가 구현한 기능을 수행하도록 additional.c 파일을 추가한다.

* 1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**



원하는 출력 결과와 함께 정상적으로 powering off 하는 모습을 볼 수 있다.