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

담당 교수 : 김영재

조 / 조원 : 20211530 김희진

개발 기간 : 23.09.23~10.07

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

pintOS는 booting과 application 실행 그리고 power off를 할 수 있는 간단한 OS라 할 수 있다. 하지만 현재 pintOS는 system call, system call handler, argument passing, user stack등이 구현되어 있지 않다. 해당 프로젝트에서는 운영체제와 pintOS 구조에 대한 이해를 바탕으로 argument passing, user memory access, system call을 구현하는 것이 목표이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

사용자가 입력한 명령어를 공백을 기준으로 parsing하고 올바른 규칙으로 stack에 넣어주었다.

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명Lib/stdio.c에 내장되어있는 함수인 hex\_dump()를 이용하여 이를 확인한다.

1. User Memory Access

user가 memory에 접근할 때, 잘못된 주소가 사용자에게 넘겨지는 상황에 대해 thread를 exit시켜준다.

잘못된 주소가 사용자에게 넘겨지는 경우는 kernel 영역의 메모리가 넘겨지거나 사용자가 잘못된 주소 그리고 mapping되지 않는 virtual address가 넘겨지는 등의 일을 말한다.

1. System Calls

사용자가 6개의 system call Halt(), exit(), exec(), wait(), read(), write() 그리고 2개의 새로운 system call Fibonacci, max\_of\_four\_int을 요청했을 때 system call handler를 통해 이들이 수행된다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

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

사용자로부터 입력 받은 명령어를 공백문자를 기준으로 parsing하고 “\0”문자를 뒤에 붙여준다. 명령어 수와 parsing된 명령어를 변수에 저장하고 이를 이용해 스택에 argument를 쌓아준다.  
스택은 위에서 아래로 자라므로 parsing된 명령어 중 가장 뒤에 있는 명령어부터 stack에 쌓아준다. 예를 들어 parsing된 명령어가 “/bin/ls”, “-l”, “foo”, “bar라면 “bar\0”부터 “foo\0”의 순서로 쌓아준다. 명령어를 쌓아준 후 워드 단위를 즉, 4의 배수단위를 맞추기 위해 null 값을 쌓아준다. 이후, argument의 address와 argument 수, return address를 차례로 쌓아준다.

* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명
    1. pintOS에서 virtual memory 개념에 대해 설명

pintOS는 user memory영역과 kernel memory영역으로 나누어져 있다. 우리가 이 메모리들에 직접적으로 접근한다면 process가 서로에게 damage를 끼칠 수 있으며 process가 kernel code를 바꾸는 등의 문제를 관리하기 어렵다.  
텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

그래서 process와 kernel code를 보호하기 위해 pintOS는 virtual memory system을 채택하여, process가 자신의 가상 메모리 영역을 가지고 전체 메모리를 차지하는 것처럼 이를 사용한다. 이러한 virtual memory도 user virtual memory(0GB~3GB)와 kernel virtual memory(3GB~4GB)로 나누어 관리된다. User영역과 kernel 영역을 나누는 경계인 3GB를 PHYS\_BASE라 한다. pintOS에서 memory unit은 4KB크기의 page이며 user program은 page directory를 해석함으로써 physical memory에 mapping 되어있는 virtual address에 접근한다. 즉, page directory에 physical memory와 virtual memory의 mapping된 쌍이 저장되어 있다.

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + 1. invalid memory access 개념

위와 같은 방식으로 memory를 관리하는 pintOS에서 잘못된 메모리 접근이 일어나는 경우는 다음과 같다. 사용자가 잘못된 포인터 즉, NULL 포인터나 PHYS\_BASE보다 큰 값 즉, 커널 메모리에 대한 포인터가 주소공간에 접근하거나 mapping이 되지 않은 virtual memory에 접근하는 경우이다.   
그러므로 우리는 이러한 경우에 대해 예외처리를 해 주어야 한다.

* + Invalid memory access를 어떻게 막을 것인지 설명

Pagedir\_get\_page()를 이용해 mapping이 되지 않은 virtual memory에 접근하는 것을 방지해준다. 또한, Memory 접근이 필요한 system call에 대한 명령이 들어왔을 때 memory접근이 valid한지 검사를 해준다. 사용자가 kernel영역을 참조하는 경우 경우 page fault가 발생할 것이므로 page fault를 관리하는 함수에서 exit(-1)를 시켜주어 이를 처리해준다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

pintOS는 process와 kernel code를 보호하기 위해 user virtual memory와 kernel virtual memory로 나누어 memory를 관리하고 있다. 그렇기에 user program은 kernel memory에 직접적으로 접근할 수 없다. 이런 상황에서, user program이 kernel’s functionality를 사용하게 하기 위해 OS가 제공한 것이 system call이다.

* + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
    - 1. Halt

shutdown\_power\_off()를 이용해 pintOS를 종료한다.

* + - 1. Exit

process termination message를 출력하고 인자로 받은 exit status를 kernel에 반환하고 현재 user program을 종료한다. Status가 0이라면 성공, 0이 아니라면 에러를 의미한다.

* + - 1. Exec

인자로 받은 file이 NULL일 때 예외 처리를 한 후, file의 이름을 가진 child process를 만든다. 새로운 process의 id를 리턴한다.

* + - 1. Wait

인자로 들어온 pid를 가진 child process가 죽을 때까지 wait하고 죽었을 때 exit status를 얻어 이를 반환한다. 반환하기 전에 process가 죽는 것을 막기위해 semaphore synchronization을 이용한다.

* + - 1. Read

인자로 받은 file descriptor가 0인 경우에 input\_getc()을 이용해 한 글자 씩 읽는다. 읽은 byte 수를 반환한다.

* + - 1. Write

인자로 받은 file descriptor가 1인 경우에 put\_buf()를 이용해 출력한다. 쓴 byte 수를 반환한다.

* + - 1. Fibonacci

인자로 받은 n번째 피보나치 수를 반환한다.

* + - 1. max\_of\_four\_int

인자로 받은 4개의 값 중 최댓값을 반환한다.

* + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명
    1. main함수에서 run\_actions(argv)가 실행된다.
    2. Run\_actions함수에서 {“run”, 2, run\_task}와 a->function(argv)에 의해서 run\_task(argv)가 실행된다.
    3. Run\_task(argv)에서 process\_wait(process\_execute(task))를 호출해주어 process\_execute(argv[1])가 실행된다.
    4. Process\_execute()에서 thread\_create 함수를 통해 파일 이름을 등록하고 start\_process가 실행되어 user program이 실행된다.
    5. Process scheduling이 실행되고 child process(user program)은   
       lib/user/entry.c에 있는 \_start함수를 통해서 실행된다.
    6. User program이 종료되면 exit(main(argc, argv))가 호출되어 종료된다. 3번의 process\_wait에 의하여 부모 process는 process\_wait에서 child process가 죽을 때까지 기다린다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

09.23~09.26 : pintOS 매뉴얼 분석과 이해, 프로젝트 파악

09.27 ~ 09.28 : argument passing 구현

09.30 ~ 10.1 : user memory access, system call(halt, exit, execute) 구현

10.02 ~ 10.04 : system call(additional, wait, read, write) 구현

10.6~10.7 : 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. Argument Passing
* Userprog/process.c의 load()의 윗부분에서 argument parsing을 위한 코드를 추가한다.
* Userprog/process.c에서 스택을 쌓기 위한 build\_stack()를 load() 아래에 추가하고 userprog/process.h에도 이 함수를 추가한다.  
  load()에서 setup\_stack()을 호출한 후, build\_stack()를 호출한다.
* Userprog/process.c의 load()에서 filesys\_open함수의 인자에 파일 이름이 들어가도록 수정해준다.
  + 1. User Memory Access
* Userprog/exception.c의 page\_fault()에 kernel memory에 접근하는 것을 막기위한 코드를 추가해주었다. 이 과정에서 함수를 사용하기 위해 필요한 헤더파일을 include한다.
* Userprog/syscall.c에서 user memory에 접근하도록 하기위한   
  check\_address()를 추가하고 userprog/syscall.h에도 해당 함수를 추가한다. Check\_address()를 작성하는 과정에서 필요한 함수를 사용하기 위해 관련된 헤더파일을 include한다.

Check\_address()는 syscall\_handler()에서 memory와 관련된 system call인 exit, execute, wait에 call 된다.

* + 1. System Calls
* Lib/syscall-nr.h에 새로운 2가지 system call(Fibonacci, max of four int)을 추가해주고 이 순서에 맞추어 userprog/systemcall.c의 syscall\_handler함수의 switch 문을 작성한다.
* Max of four int를 위해서 lib/user/syscall.c에서 syscall4()를 define하고 Fibonacci()와 max\_of\_four\_int()함수를 추가하여 system call API를 정의한다.  
  lib/user/syscall.h에도 해당 함수를 추가한다.
* Userprog/syscall.h에서 lib/user/syscall.h를 include한다.
* Userprog/syscall.c의 syscall\_handler() 아래에서 system call을 실행하기 위한 syscall\_halt(), syscall\_exit(), syscall\_exec(), syscall\_wait(),   
  systcall\_read(), syscall\_write(), syscall\_fibonacci(), syscall\_max\_of\_four\_int()를 작성하고 userprog/syscall.h에 함수들을 추가한다.  
  이 과정에서 필요한 header파일을 userprog/syscall.c에서 include한다.
* Syscall\_exec()와 syscall\_wait() 함수는 process\_execute(), process\_wait()함수를 호출하므로 userprog/process.c에서 두가지 함수에 코드를 추가하고 수정한다.
* Wait를 구현하는 과정에서 threads/thread.h의 thread 구조체에 코드를 추가하고 threads/thread.c의 init\_thread()에 코드를 추가한다.   
  이 과정에서 필요한 헤더파일들을 threads/thread.h에 include 한다.
* 새로운 두가지 system call을 실행할 수 있도록 하기 위해서 examples에 additional.c를 추가하고 Makefile을 수정한다.

1. **연구 결과**
   1. **Flow Chart**

**II. B. 개발 내용에 대한 Flow Chart를 작성**

1. 아래 flow char에 대한 이해를 돕기 위해 전체적인 작동에 대한 flow chart

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Argument Passing

텍스트, 폰트, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 도표, 원, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 도표, 텍스트, 기술 도면, 스케치이(가) 표시된 사진

   자동 생성된 설명System 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

* userprog/process.c

load()에서 parsing을 위해 다음과 같은 코드를 추가하였다. Argc는 명령어 개수를 저장하는 int변수이며 argv[]는 parsing한 명령어를 추가하는 int 배열이다. Strtok\_r()와 while문을 통해서 공백문자를 기준으로 명령어를 parsing해 주었다. Stack에 쌓는 과정에서 명령어+”\0”형태로 쌓아야하는데 strtok\_r함수를 이용하면 NULL문자가 parsing된 명령어 뒤에 붙게된다.

파일을 여는 filesys\_open()은 인자로 파일이름을 받기 때문에 parsing한 명령어의 가장 첫번째 명령어를 인자로 넣어주었다. 이 방법으로 load() fail문제를 해결했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

명령어를 stack에 쌓기위해 build\_stack이라는 함수를 추가로 정의해 주었다. 이 함수는 load()에서 set\_up stack()을 실행한 후에 call해주었고 stack의 주소를 가리키는 esp와 명령어 개수와 명령어를 저장하는 argc, argv를 인자로 넘겨주었다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

다음은 build\_stack() 함수의 코드이다. Parsing된 명령어 중 뒤의 명령어부터 stack에 쌓아주어야 하기 때문에 argc-1부터 0까지 for문을 돌며 memcpy()를 이용해 stack에 명령어를 쌓아준다. 이때 word\_align변수는 word단위로 주솟값을 맞추기 위한 변수이므로 parsing된 명령어들의 길이를 for문을 돌며 저장해준다(null문자 포함). adrs[]는 명령어들의 주솟값을 저장하는 배열이므로 for문을 돌며 parsing된 명령어의 주솟값을 저장해준다. 저장된 word\_align변수를 word 단위인 4로 나눈 나머지만큼 null 값을 채워준다. 그 후, null값을 쌓고 adrs[]를 이용해 명령어들의 address를 쌓고 argc, return address를 차례로 stack에 쌓아준다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* userprog/process.h

추가한 build\_stack()을 헤더파일에 추가한다.

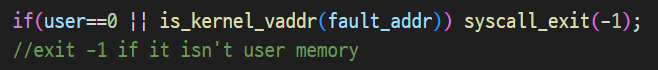
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

* Userprog/exception.c

Threads/vaddr.h에 정의되어있는 is\_kernel\_vaddr()과 page\_fault()의 flag인 user변수를 이용해서 kernel memory에 접근했을 때 syscall\_exit(-1)를 call하여 exit 한다.



Is\_kernel\_vaddr()을 이용하기위한 threads/vaddr.h과 syscall\_exit()를 이용하기 위한 userprog/syscall.h를 참조한다.

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

* userprog/syscall.c

threads/vaddr.h에 정의되어 있는 is\_user\_vaddr()을 이용해 valid한 user memory에 접근하고 있는지 확인해주는 check\_address()를 정의하여 사용했다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

해당 함수는 syscall\_handler()에서 user memory인지 확인이 필요한 system call인 exit, execute, wait에 사용된다.

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명** Is\_user\_vaddr()을 이용하기 위해 threads/vaddr.h를 참조한다**.**

1. System Calls

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

* userprog/syscall.c

Lib/syscall-nr.h의 순서를 참고하여 syscall\_handler()에서 switch문을 작성한다. 이때 switch문의 인자로 들어오는 \*(uint32\_t\*)(f->esp)는 syscall number이며 이에 따라 어떤 system call인지 판단할 수 있다. switch문은 명령으로 들어온 system call이 무엇인지 판단하여 case별로 이를 실행할 함수를 call한다.

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

switch문에서 사용되는 system call을 실행하기 위한 함수들은   
syscall\_handler() 아래에 작성했다.

Shutdown\_power\_off()를 이용해 halt가 명령으로 들어왔을 때 처리를 해주었다**.**

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명현재 thread의 exit\_status로 함수의 인자로 들어온 status를 저장해주어 kernel로 반환한다. termination message를 출력한 뒤 thread\_exit()를 호출하여 exit해준다. Exit\_status는 thread구조체에 추가한 요소이며 변수명 그대로 process의 exit 상태를 저장하는 변수이다.

Execute와 wait는 각각 process\_execute(), process\_wait()를 call해주어 처리했으며 이 함수들은 userprog/process.c에서 설명할 것이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이번 project에서는 stdin에 대해서만 read를 구현해줄 것이므로 syscall\_read()에서는 인자로 받은 file descriptor가 0(STDIN)이 아닌 상황에 -1을 반환한다. Fd가 0인 경우에는 input\_getc()를 이용해 한글자씩 읽어주고 읽어준 글자 수를 반환한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이번 project에서는 stdout에 대해서만 write를 구현해줄 것이므로 syscall\_write함수에서는 인자로 받은 file descriptor가 1(STDOUT)이 아닌 상황에 -1을 반환한다. Fd가 1인 경우에는 putbuf()를 이용해 buffer에 write해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Syscall\_exit()함수에서 thread 구조체의 exit\_status에 값을 저장해 주었다.   
이를 위해 threads/thread.h를 include했으며 syscall\_halt()에서   
Shutdown\_power\_off()를 이용하기 위해 devices/shutdown.h를 include한다. 또한, syscall\_exec()과 syscall\_wait()에서 userprog/process.h에 있는 함수를 return 하므로 이도 참조한다.

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**-** userprog/syscall.h

새롭게 정의한 syscall 명령어 함수들을 헤더파일에 추가해 주었으며 lib/user/syscall.h를 include해서 system call 명령이 들어왔을 때   
syscall\_handler가 올바르게 동작될 수 있도록 했다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**-** userprog/process.c

Process\_execute()에서 인자로 주어진 file을 open했을 때 NULL값인 경우에 대해 예외처리를 해주었다. 파일이름을 얻기 위해서 들어온 file name을 공백문자를 기준으로 잘라서 사용하였고 thread\_create()에서도 파일이름이 필요하므로 thread\_create()의 첫번째 인자에도 이를 사용했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Process\_wait()에서는 현재 thread의 child list를 돌며 인자로 받은 pid를 가진 child thread를 찾아준다. s\_child semaphore는 0으로 초기화되어 있는데 그렇기 때문에 child process가 종료되어 process\_exit()에서 s\_child를 up시킬 때까지 parent thread는 wait한다. Process\_exit()에서 s\_child가 up된 이후 proess\_wait()에서 s\_child를 down시켜주며 list\_remove를 통해서 child process가 없어졌음을 반영한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명이렇게만 코드를 짰을 때, 메모리가 랜덤으로 남아있을 수도 없을 수도 있기 때문에 list\_remove를 수행하는 과정에서 에러가 떴다. 이를 memory에 대한 semaphore인 s\_mem을 추가하여 해결했다. process\_exit()에서 s\_child를 up시킨 후, s\_mem을 down시켜 process\_wait()의 과정이 끝날 때까지 memory를 lock시켰다. Process\_wait()에서 list\_remove의 과정이 끝난 후, s\_mem을 up해주어 memory를 다시 unlock해주고 exit\_status를 반환한다.

폰트, 그래픽, 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- threads/thread.h

앞서 설명한 process\_wait()과 syscall\_exit(), process\_exit()에서 사용한 thread 구조체의 요소들은 thread.h의 thread 구조체에 추가했다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

Semaphore를 사용하기 위해 threads/synch.h를 참조했다.



* Threads/thread.c

Init\_thread()에서 thread 구조체에 새롭게 추가해준 요소들에 대한 초기화를 진행한다. Process\_wait()의 설명과정에서 언급한 대로 semaphore는 0으로 초기화한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Additional System calls

**새로운 시스템 콜(fibonacci, max\_of\_four\_int)을 구현하기 위해 수정하거나 작성한 코드에 대해 서술**

* Lib/syscall-nr.h

새롭게 정의할 systemcall SYS\_FIB, SYS\_MAX를 정의한다.

폰트, 텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

* Lib/user/syscall.c

Max\_of\_four\_int()에 필요한 인자로 4개가 들어오는 system call에 대한 syscall4를 define하고 새로운 system call에 대한 system call API를 정의한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Lib/user/syscall.h

새롭게 정의한 함수들을 추가한다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

* Userprog/syscall.c, syscall.h

명령을 실행할 함수를 구현하고 이를 header파일에도 추가한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명**

* examples/additional.c

텍스트, 스크린샷, 멀티미디어 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명Fibonacci()와 max\_of\_four\_int() system call을 실행하기 위해서 addition.c파일을 추가하여 작성한다.

-examples/Makefile

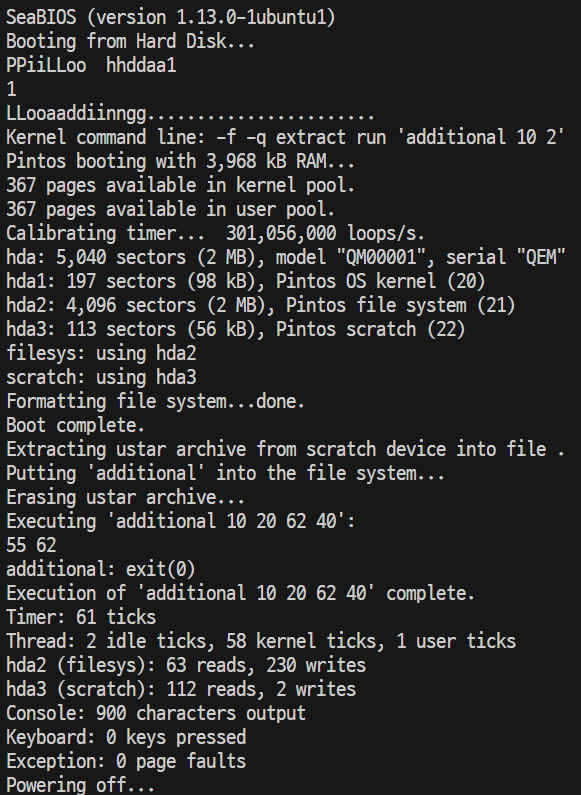
새롭게 추가한 additional.c를 실행하기 위해서 makefile을 수정한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

1. **시험 및 평가 내용**

* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**

****