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

담당 교수 : 김영재

조 / 조원 : 20191626 이용욱

개발 기간 : 2023.09.22 ~ 2023.10.07

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

skeleton 코드의 핀토스는 user program이 system call function을 호출하면, 일련의 system call handler 함수의 실행 없이 즉시 return하는 구조로 설계되어 있다. 그러므로 이 프로젝트는 kernel의 system call handler function을 작성하여 kernel의 system call 기능을 설계하는 것이다.

이를 위해 stdin 으로 입력받은 명령어 라인을 공백 즉,“ “단위로 parsing하는 것과 이 parsing한 결과를 user program에 넘겨 주어 해당 user program이 system call을 시행할 때 필요한 argument들을 정상적으로 kernel에 제공하도록 하는것이 목표이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

parsing 처리를 완료한 argument 들을 스페이스 단위로 parsing 한 후에 , user program의 stack에 쌓는다. 특정 program이 system call function을 call했을 때, kernel은 system call handler가 필요한 매개 인자들에 접근하도록 한다. kernel은 이 때 interrupt frame을 통해 user program의 가상 메모리 esp를 받아와서 input을 이용하여 system call handler function을 실행하도록 한다.

1. User Memory Access

여기에서, User program은 실제 물리적 메모리에 접근하는 것이 아니라, cpu의 가상화를 위해 가상 메모리에 접근하여 값을 저장한다. 따라서 virtual address를 physical address로 변환해 줄 수 있어야하며, 잘못된 메모리 주소에의 접근을 throw해야 한다. 그러므로 kernel의 고유 memory 영역이 침범되지 않도록 user memory access를 구현한다.

1. System Calls

user program은 kernel 고유의 함수들을 직접 수행하면 robustness가 위협될 수 있다. 따라서 user program이 특정 명령을 요청하면, kernel이 대신 수행하여 결과를 user program에게 전달하도록 해야 한다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing

setup\_stack() 함수 이후에, esp =PHYS\_BASE로 초기화한다. 그 이후 argument들을 x86 convention에 따라 쌓는다. 이 때 공백 단위로 자른 argument들의 길이만큼 esp를 감소하고, memcpy()함수와, 다른 esp 연산을 통해, esp pointer가 가리키는 주소에서부터 메모리에 적재한다. 그리고 위의 과정이 완료되면, 필요할 시 word alignment 또한 완료한다. 마지막으로 x86 convention에 따라 esp값, 각 인자들의 시작 주소, 각 인자들의 개수, NULL pointer 까지 스택에 적재한다.

* User Memory Access

PintOS에서는 kernel 영역의 memory와 user program의 memory가 완전히 분리되어 있어야한다. 이 규율이 어긋나면, user program의 wrong access로 인해 심각한 에러가 발생할 수 있다. 따라서 user program이 잘못된 메모리 영역에 접근을 시도할 경우 kernel은 user program을 halt & exit 시켜야 한다. 이러한 invalid memory 접근에서는 잘못하면 kernel addrress space를 접근하게 된다. 그러므로 매번의 system call 에서 정의한 validation\_check 함수를 호출해서 위의 경우들을 검출하고, 방지한다.

* System Calls

시스템 콜은 user program이 kernel의 메모리 영역에 저장된 안전하고 유용한 값에 접근할 수 있도록 한다. 이를 통해 user program의 활동 반경에 제약을 둠으로써 robust 한 software를 설계할 수 있게 된다. 해당 프로젝트에서 개발한 시스템 콜들은 아래와 같다.

먼저 변수 sys\_num 선언하여 알맞은 syscall number을 할당한다. 그 후에 switch문으로 각각의 number에 맞는 기능을 구현한다. system call 함수의 구현에서 invalid memory access를 막기 위한 코드도 삽입한다.

1) SYS\_HALT

pintos를 중단하는 시스템 콜이다. halt()에서 shutdown\_power\_off() 함수를 사용한다.

2) SYS\_EXIT

현재 실행중인 user program을 종료한다. 인자 하나를 받는데 exit staturs를 return값으로 받는다. exit함수에서 thread\_exit() 함수를 사용한다.

3) SYS\_EXEC

child process를 만드는 시스템 콜이다. exec() 함수에서 process\_execute() 함수를 사용한다. thread id를 반환한다.

4) SYS\_WAIT

parent thread가 child thread보다 먼저 끝나지 않도록 wait 함수를 이용한다. exit status 를 반환한다.

5) SYS\_READ

이 프로젝트에서는 STDIN으로 부터 입력을 받는다. read 함수에서 구현하는데 이 때, input\_getc() 함수를 사용한다.

6) SYS\_WRITE

이 프로젝트에서는 STDOUT에 대해서만 다룬다. write 함수를 통해 구현하고, write 함수에서 putbuf() 함수를 사용하여 출력한다.

7) SYS\_FIBO

입력받은 값에 대한 피보나치 수를 구하기 위한 시스템 콜이다. fibonacci 함수에서 여러 변수들과 for 문을 사용하여 피보나치 수를 구한다.

8) SYS\_MAX

max\_of\_four\_int를 위한 시스템 콜이다. max\_of\_four\_int 함수를 call하여 구현한다.

user 레벨에서 시스템 콜 api를 요청하면, syscall.c에서 정의되어있는 알맞은 시스템 콜 함수가 호출된다. 이 때 interrupt 가 발생하는데, 이것이 kernel code 가 실행되도록 하고, argument들을 통해서 system call handler 가 실행된다. 실행 이후 반환 값은 eax reg에 저장되고, 다시 userprogram에 전달된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **9**

9월 22일 ~ 25일 : argument parsing 구현

9월 26일~ 28일: passing 구현

10월 1일 ~10월 7일 : system call구현

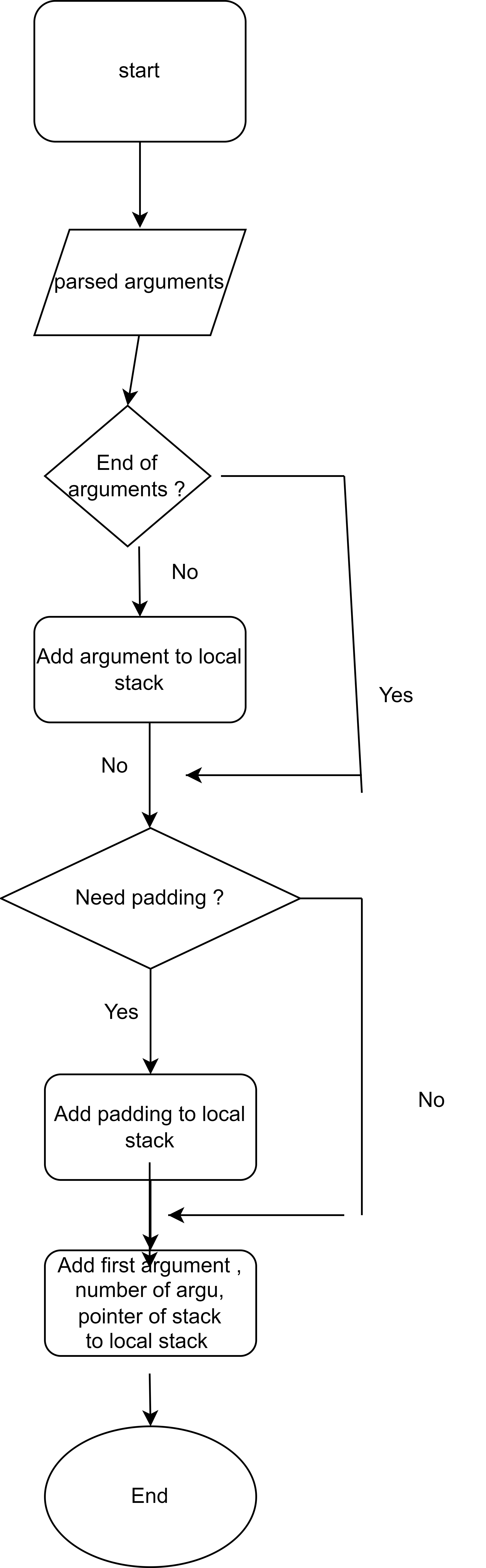
* 1. **개발 방법**
* **II. B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

우선 thread.h에서 thread 구조체에 여러 요소를 추가하여 재정의한다. 세마포어 요소들을 추가하여 code flow 가 올바르게 작동되도록 하고, 그 thread의 child list를 추가해준다. 그리고 exit\_status / fl 를 추가한다. setup\_stack() 함수 이후에 passing을 위한 코드를 추가하며, process\_execute, process\_wait(), process\_exit()과 같이 process의 실행과 종료와 관련된 함수들을 변경된 thread 구조체에 알맞게 코드를 수정해준다. 그 후 userprog / syscall.c 에 각 system call에 알맞은 system call handler 들을 추가해준다.

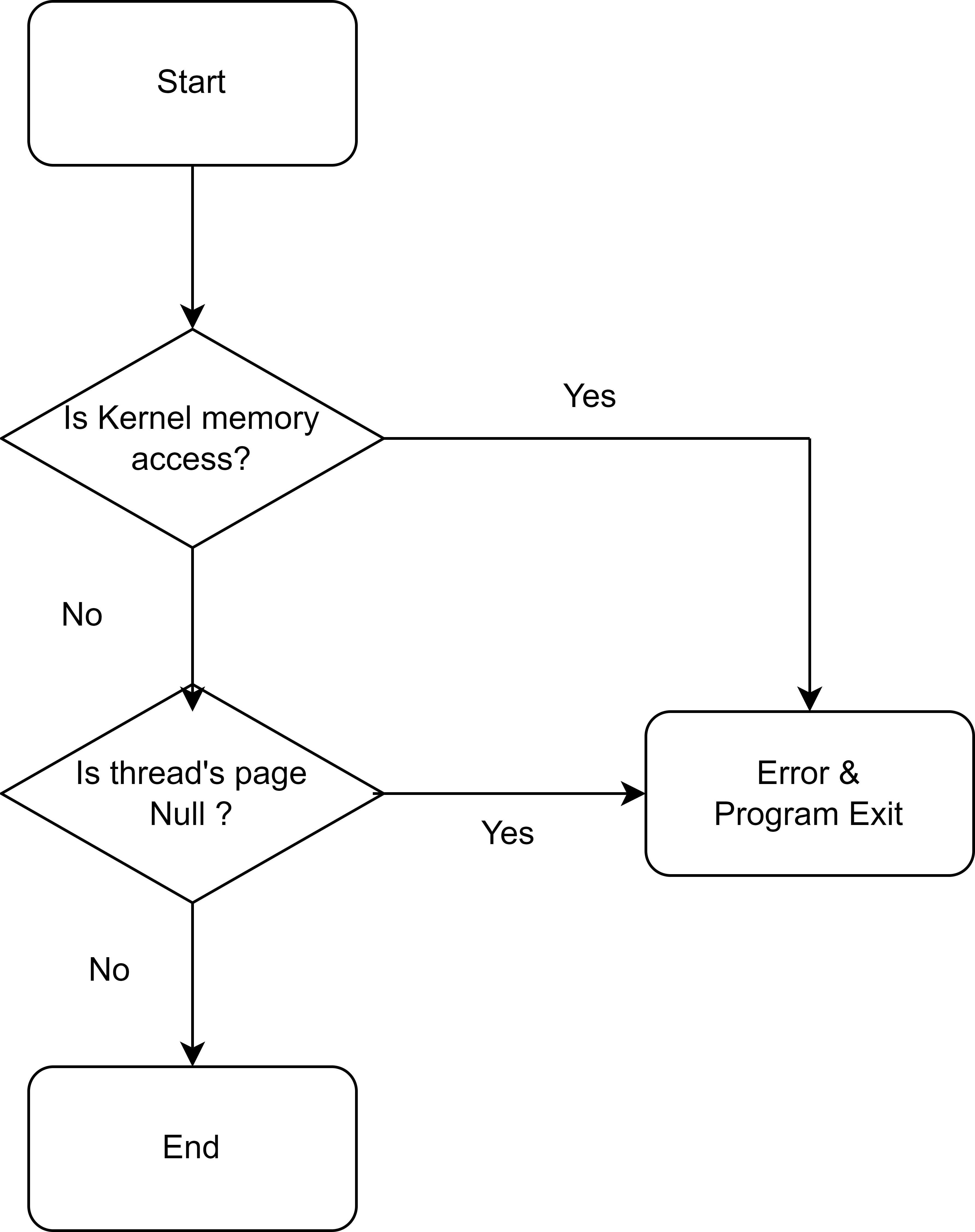
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

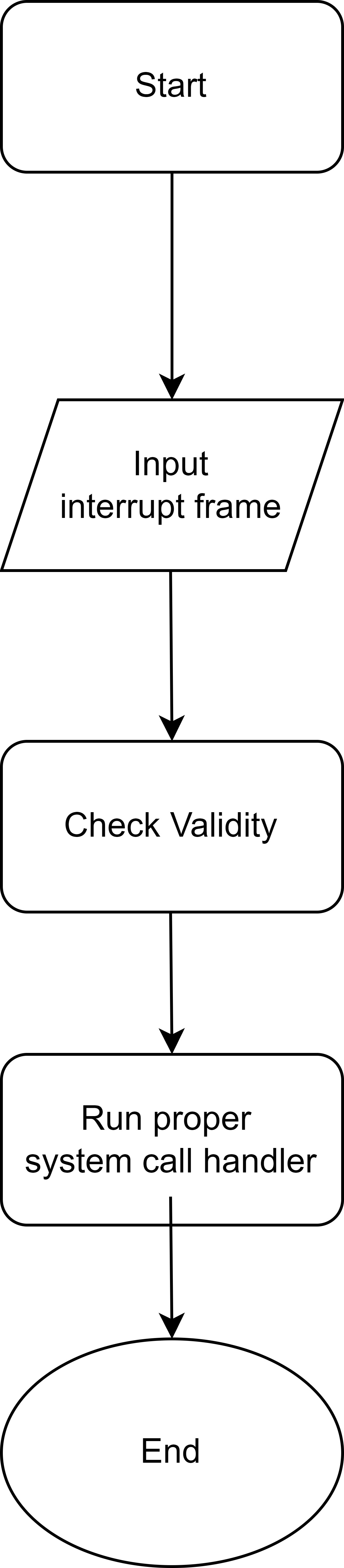
1. Argument Passing



1. User Memory Access



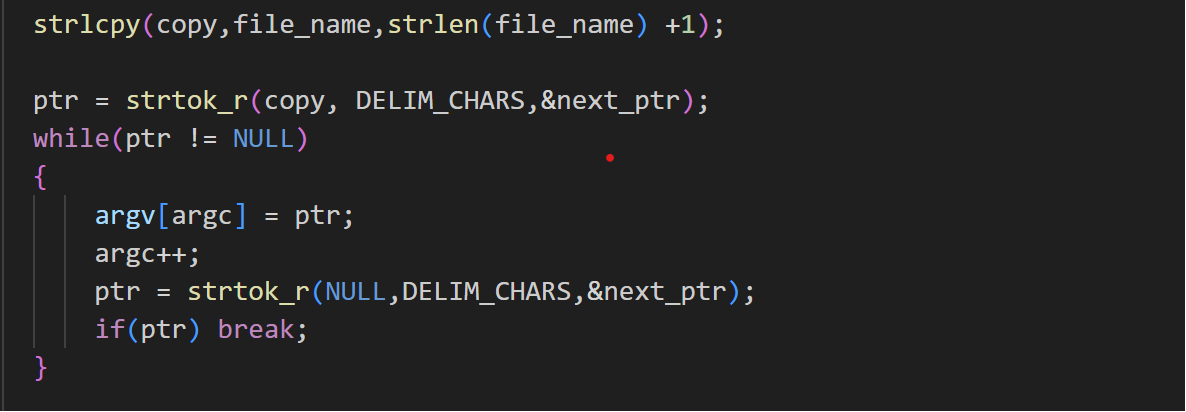
1. System Calls



* 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

우선 argument들을 공백 단위로 나누어 각각 구분하기 위해서 strtok\_r() 함수를 사용한다. 그리고 stack에 적재할 때는 esp 값을 규칙에 맞게 감소시킨 후 memcpy() 또는 memset() 함수를 이용하거나 일련의 스택연산으로, user program의 local stack에 x86 convention를 따라서 argument들을 저장한다.



위는 file\_name을 copy에 복사하고 strtok\_r 함수를 이용해서 argc, argv를 저장하는 코드 부분이다.

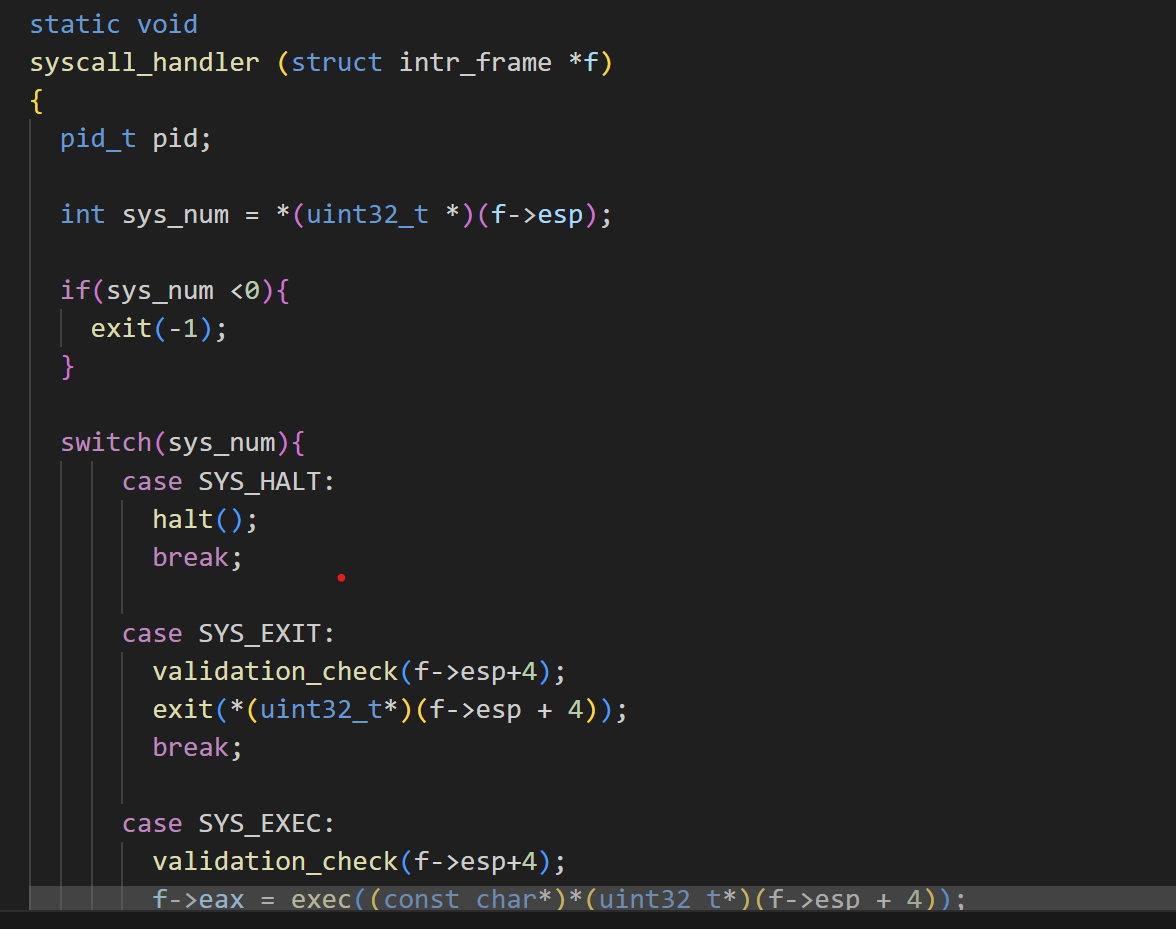
1. User Memory Access

system call 이 발생했을 때 , 직접 정의한 validation\_check () 함수를 call해서 invalid user memory access를 검출한다. 이 때, validation\_check()에서는 is\_user\_vaddr(), is\_kernel\_vaddr() 함수를 사용하여 invalid memory access 를 검출하고, 이경우 exit(-1)로 강제종료 시킨다.

1. System Calls

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

system call 이 전달 받은 interrupt frame으로 해당 user program의 esp 에 접근한다. esp의 값들을 통해 sys\_num을 확인하고 해당 시스템 콜에 알맞은 handler를 실행한다. 또한 validation\_check를 통해서 user memory access 를 확인하고, 이후 앞에서 정의한 system call handler가 수행해야할 동작에 따라서 코드를 작성한다. 세마포어가 필요한 부분에는 적절히 사용해서 안정된 동작을 추구한다.

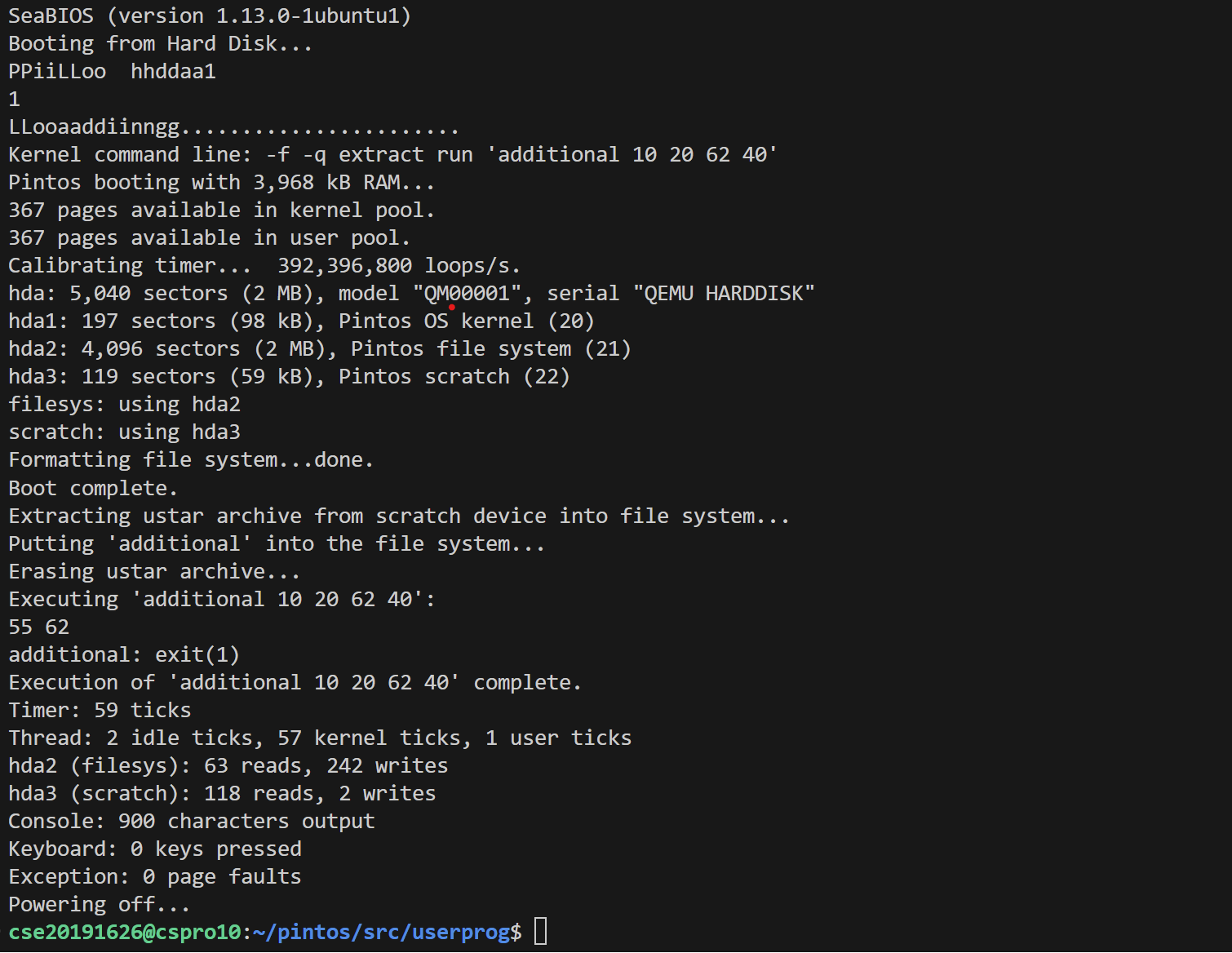


위는 switch 문을 이용해서 sys\_num에 따른 처리 코드이다.

1. Additional System calls

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

fibonacci는 1개의 number을 입력받아서 피보나치의 점화식 f(n) = f(n-1) + f(n-2) 를 이용하여 결과를 구한다. 마찬가지로 max\_of\_four\_int 에서도 max 값을 리턴하는 구조를 구현한다. 그리고 syscall-nr.h에서 새로운 2개의 system call number을 지정하고, userprog에서 syscall.c에 두 개의 system call handler를 추가한다. 이 때 각 시스템 콜에 맞춰서 user memory access 를 확인하고, return 값을 eax register에 반환한다.

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