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

담당 교수 : 박성용 교수님

조 / 조원 : 20191597 서윤혁

개발 기간 : 3주

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

pintos os에서 user program을 구현하는 프로젝트였다.

해당 프로젝트에서 주요적으로 구현해야했던 내용은 argument parsing, argument passing, syscall\_handler 및 syscall을 구현하는게 주였다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

우선 user의 command를 “ “단위로 split해주어 argument 배열에 넣어주고나서, 스택에 해당 argv를 "3.5 80x86 Calling Convention” 에 맞추어서 쌓아주었다.

1. User Memory Access

syscall.c 파일에서 kernel이 유저의 메모리 영역에 접근할 때 해당영역이 유저의 메모리 영역인 지 체크하는 로직을 넣어주었다.

1. System Calls

system call 같은 경우 syscall.c 파일의 syscall\_handler() function 안에 구현을 하였다. 우 선 syscall number에 따라서 system call들을 구분하였고, 그리고나서 각 system call들이 알 맞는 동작을 하는 메서드들을 추가하여 해당 system call이 들어왔을 때 알맞는 동작을 하도록 구현했다.

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

우선 핀토스의 "3.5 80x86 Calling Convention”을 따라 스택을 쌓아주었다.

stack이 메모리에서 아래로 쌓인다는 것을 인지하고 시작해야한다.

그리고 나서 convention을 보고 차례로 argument 배열에 저장되어 있는 값들을 꺼내어 뒤에서부터 넣어주고 word-alignment를 맞춰준다. 32bit이므로 4바이트로 alignment가 된다.

이때 argument 배열안에 있는 값들을 esp주소를 넣어 업데이트를 해준다. 왜냐하면 뒤에서 스택에 esp의 주소를 넣어주어야 하기 때문에 미리 업데이트를 해놓는다.

그리고 스택에 null값을 넣어주고, 이제 아까 argument 배열에 저장해두었던 주소값들을 다시 right-to-left 방향으로 스택에 넣어준다. 그리고 스택에 argument 시작 주소를 저장해두고, 그리고 argument 개수를 저장하고 다시 null값을 저장해주면 된다.

* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명

핀토스에서는 user 영역의 메모리와 kernel 영역의 메모리가 나누어진다.

이때 user가 kernel 영역의 메모리에 접근하게되는 경우나, user program에서 null pointer를 넘겨주는 경우, virtual memory에 매핑이 되어있지 않은 경우 모두 invalid memory access가 된다.

* + Invalid memory access를 어떻게 막을 것인지 설명

kernel 영역에서 user program에서 넘어오는 pointer들을 validation 해보면 된다.

user program에서 pointer가 invalid한 경우에 page\_fault가 일어날 수 있는데, 이때 exception handling을 통해 user program에서 해당 invalid memory access를 막을 수 있다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

시스템 콜은 user의 악의적인 호출이나 잘못된 호출을 방지하기 위해서 kernel 모드에서만 수행할 수 있도록 만들어 놓은 것. 따라서 시스템 콜을 호출할 때 user는 반드시 kernel로 control을 위임하여야 한다.

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

halt : pintos os를 종료하기 위한 system call. shutdown\_power\_off() 함수를 통해 종료할 수 있다.

exit: 현재 작동하는 process를 종료시키기 위한 system call. Kernel의 status를 return 한 다. 정상적으로 종료시 0을 return.

exec: exec는 child process를 생성할 때 사용하는 system call.

wait: child process가 종료될 때까지 기다렸다가 child process가 종료되면 child process가 종료될 때의 exit status를 가져와서 해당 status를 반환한다.

read: 열려있는 파일 또는 STDIN에서 데이터를 읽는 system call.

write: 열려있는 파일 또는 STDOUT에 데이터를 쓰는 system call.

* + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

System call 호출 시 argument 개수에 따라서 syscallN이 불려지고, 해당 함수에서 argument를 stack에 push후 int $0x30 instruction 으로 interrupt를 발생시킨다. 그 후 syscall\_init함수에서 syscall\_handler를 호출하고 syscall\_handler에서는 stack에서 syscall num을 보고 알맞은 sycall을 수행하는 함수를 호출한다. 여기서 return값을 eax 레지스터에 저장해서 다시 user가 사용할 수 있게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

9/28: Argument parsing, Argument passing 구현

9/29: Syscall handler 구현, Additional System call 구현

* 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)**

Argument parsing & Argument passing:

Process.c 파일의 load함수에서 file\_name을 parsing 한 후, esp에 "3.5 80x86 Calling Convention”에 맞춰 push 해준다.

System call:

Syscall.c의 syscall\_handler 함수에 스택에 저장되어 있는 syscall number를 이용해 system call들을 구분하고 해당 system call을 수행하는 함수들을 작성한다.

Wait system call을 구현할 때 thread.h에 정의된 thread 구조체의 수정이 필요하다. Wait 구현에 필요한 세마포어나 exit\_status, child 접근을 위한 child\_list, child\_elem\_list를 추가 해주었다.

User Memory Access:

Syscall\_handler 함수에서 syscall 함수를 수행하기 전에 valid한 접근인지 체크하는 함수를 실행시켜서 validation을 해준다.

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

1. System Calls

도표, 라인, 평면도, 평행이(가) 표시된 사진

자동 생성된 설명

* 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 process.c의 load함수에서 넘어온 file\_name에서 strtok\_r을 이용하여 parsing을 먼저 해주고 argv 배열에 담아주었다. 그리고 총 argument가 몇인지 argc를 이용해 측정하였다. 그리고 해당 argument를 stack에 push를 해주어야 하는데 80x86 convention에 맞추어서 stack에 push를 해준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
해당 코드는 stack을 push 하는 과정인데, 우선 argv에 저장되어 있는 argument들을 뒤에서부터 push를 해주고, argv 배열에 다시 stack의 주소를 저장을 한다.

그리고 32bit이므로 4byte 단위로 word-alignment를 해주고 나서 null 값을 저장해준다.

그리고 argv에 방금 저장한 스택의 주소를 다시 뒤에서부터 push를 해주고,

argv의 시작주소를 저장해주고 argument 개수인 argc를 저장해주고 null 값을 저장해준다.

1. User Memory Access

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
위의 코드는 syscall\_handler에서 user memory access를 정확히 했는지 체크하는 로직을 작성한 코드이다. 각 switch문에 is\_user\_vaddr을 통해서 해당 스택의 주소가 user memory에 있는지 체크하는 함수인데 vaddr.h에 있는 함수를 사용하였다.

만일 is\_user\_vaddr이 false라면 잘못된 접근이기에 exit(-1)을 호출해서 프로세슬르 비정상 종료시켜준다.

1. System Calls

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

Halt:

Halt system call이 들어오면 shutdown\_power\_off() 함수를 통해서 os를 종료시킨다.

Exit:  
exit system call이 들어오면 memory address validation을 한 후

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명|  
위의 함수를 태우는데 현재 thread의 이름과 상태를 출력하고 thread\_exit()을 시킨다.

Wait:  
wait system call이 들어오면 memory address validation 후  
폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명  
해당 함수를 통해 process\_wait를 호출하게 되는데,  
  
텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

process\_wait 함수에서는 현재 thread를 가져와서 현재 thread의 child들을 for문을 돌면서 child thread id와 argument로 넘어온 child thread id를 비교한다.

만약 찾지 못했으면 -1을 return하고, 찾았으면 우선 sema\_down 이용해서 자식 프로세스가 sema\_up을 해줄 때 까지 대기하게 하였고, 자식 프로세스가 process\_exit()을 통해 exit을 하면 sema\_up을 호출하게 되어 wait()를 호출한 프로세스가 다시 작업을 수행하면서 자식 프로세스의 exit\_status와 자식프로세스의 메모리를 회수하게 한다. 그리고 자식의 exit status를 리턴한다. 여기서 free\_sema라는 세마포어가 하나 더 사용되는데 이는 자식 프로세스의 메모리를 회수할 때까지 exit하지 못하도록 한다.

Exec:

exec system call이 들어오면 memory address validation을 한 후  
process\_execute()를 실행시킨다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

process\_execute에서는 넘어온 file\_name을 파싱해주고 해당 file\_name이 열리는 파일인지 체크하는 filesys\_open 함수를 통해 error handling을 해준 후 thread\_create할때 parsing한 file\_name을 넘기도록 한다. 생성한 thread id를 리턴한다.

Read:

read system call이 들어오면 user memory인지 체크하고 read 함수를 호출한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Read 함수에서는 STDIN으로 들어오는 유저 입력을 size만큼 input\_getc()함수로 읽고 size를 리턴한다.

Write:

Write system call이 들어오면 user memory 영역인지 체크하고 write 함수를 호출한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

write함수에서는 STDOUT에 putbuf함수를 통해 size만큼 쓰고, size를 리턴한다.

1. Additional System calls

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

우선 lib/user/syscall.c 파일에 fibonacci, max\_of\_four\_int 함수를 선언해주고, max\_of\_four\_int 함수의 경우에는 argument를 4개를 받는데, syscall3까지 구현이 안되어 있으므로 syscall4를 추가해주었다.

그리고 나서 lib/syscall-nr.h 파일에 새로운 시스템콜에 대해서 syscall num을 정의하기 위해 추가 해주었다.

마지막으로 실제 코드가 돌아갈 userprog/syscall.c 파일에 우선 switch문에 sys\_fibo와 sys\_max를 추가하여 sys\_num으로 해당 system call을 타도록 해주었고, 해당 로직을 각각 fibonacci(), max\_of\_four\_int() 함수에 구현 해주었다.

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

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**