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

담당 교수 :문의현

학번 / 이름 :20192021 / 강유석

개발 기간 :10월 6일 ~ 11월 1일

1. **개발 목표**

현재 Os의 명령어를 입력하면 그냥 단순히 문자열로 인식하는 수준이다.

입력한 명령어를 인식하고 시스템콜을 호출해 kernel 안에 있는 함수 API를 사용해

User 모드 에서도 kernel함수를 사용하여야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

입력한 명령어를 공백 기준으로 parsing하고 user stack에 쌓는 과정이다.

해당 스택을 제대로 쌓을 경우 system call을 부를 수 있다.

1. User Memory Access

User는 user공간의 메모리만 접근해야 한다. Kernel 메모리를 접근하면 안된다.

이를 방지하기 위해 kernel 메모리를 접근하면 process를 exit 하는 함수를 구현한다.

1. System Calls

User 모드에서 시스템 콜을 호출하면 kernel 모드에 있는 함수를 호출해 다시 user 모드에서 사용하게 끔 하는 과정이다.

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

Process\_execute()의 인자로 들어온 file\_name을 공백기준으로 parsing 한다.

그 다음 load() 함수 내에서 스택 포인터 esp를 활용해 한 칸당 -4를 해주며

차례대로 쌓아준다.

예를 들어 명령어가 ‘echo x’ 라면

첫번째로 ‘x’ 가 들어오고 해당 위치의 메모리 주소를 다시

Argv에 저장해 준다.

두번째로 ‘echo’가 들어오고 해당위치의 메모리 주소를 다시

Argv에 저장해준다.

현재 esp가 4의 배수이면 좀더 편하므로 4의 배수인지 확인 하고

4의 배수가 아니면 4의 배수가 될때 까지 esp를 내려준다.

그 다음 null을 저장하고 위에서 저장한 메모리 주소를

차례대로 스택에 쌓아준다.

그 다음 argv의 메모리를 쌓아주고 argc를 쌓아주고 마지막으로 return address인 0을 쌓아 준다.

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

Mapping 되지 않은 가상 메모리 공간이다.

* + Invalid memory access를 어떻게 막을 것인지 설명

../threads/vaddr.h 에 있는 is\_user\_vaddr() 를 사용해 현재 스택포인트가

Mapping되지 않은 공간을 가리키면 바로 exit 할 수 있게 하였다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

시스템 콜을 사용하면 kernel 내에 있는 함수를 안전하게 사용할 수 있다.

물론 user의 함수는 사용하기 편하다는 장점이 있지만 외부의 많이 노출될 수 있어 보안 상에서 위험할 수 있다.

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

halt() : 핀토스 함수인 shutdown\_power\_off() 를 사용해 구현하였다.

exit() : 현재 thread의 exit status를 저장하고 thread\_exit() 를 호출한다.

exec() : 인자로 받은 file을 process\_execute()의 인자로 넘겨준다.

wait() : 현재 thread의 pid를 인자로 process\_wait()으로 넘겨준다.

write() : fd가 1이면 putbuf() 함수를 호출해 구현한다.

read() : fd가 0이면 size만큼 input\_getc() 가 ‘\0’ 일 때 까지 i를 증가해주고

i를 return해준다.

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

User에서 시스템 콜 함수를 호출하게 되면 intr\_handler를 통해 syscall\_handler를 호출한다.

여기서 user스택의 esp를 kernel에 넘겨준다.

Kernel에서는 esp를 기준으로 해당 시스템 콜 번호를 보고 kernel apl를 실행 시키고 return 값을 eax에 담아 user mode로 return 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

10월 6일 부터 10월 13일 까지 해당 프로젝트의 내용을 이해하고,

Argument passing을 진행 하였다.

중간고사 기간으로 10월 26일 까지 진행을 못하다가 10월 27일 부터 memory access를 구현하였다.

10월 30일에 시스템 콜을 구현하였다. 시스템 콜중 halt()만 제대로 구현하고

나머지 시스템 콜은 다른 함수와의 연계로 진행하지 못하였다.

10월 31일에 모든 시스템 콜을 구현하였다. 하지만 시스템콜 핸들러의 오류로

인해 제출하지 못하였다.

11월 1일 모든 시스템 콜과 시스템 콜 핸들러가 정상적으로 작동하고

Make check를 통해 확인하였다.

* 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)**

src/userprog/process.c에 load() 함수내에 argument passing을 하는 소스코드를 추가한다.

src/userprog/syscall.c에 syscall\_handler() 에서 각 시스템 콜을 추가해준다.

시스템 콜 중 wait을 구현하는 과정에서 src/userprog/process.c의 process\_wait()을 추가해준다. 추가하는 과정에서 semaphore가 사용되기 때문에

threads/thread.h의 thread 구조체에 semaphore를 추가해준다.

또한 시스템 콜중 exit 하는 과정에서 현재 thread의 exit\_status를 저장해 줘야 하므로 thread 구조체 내에 exit\_status를 추가해 준다.

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing
2. User Memory Access
3. 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

텍스트이(가) 표시된 사진

자동 생성된 설명

스택을 쌓는 과정이다. 처음에 스택을 쌓는 과정에서 argv의 메모리를 쌓는 과정에서

애를 먹었다.

해결 과정은 처음에 argv의 value값을 스택에 쌓으면서 동시에 해당 위치의 메모리 값을 덮어쓰기로 넣었다.

1. User Memory Access

텍스트이(가) 표시된 사진

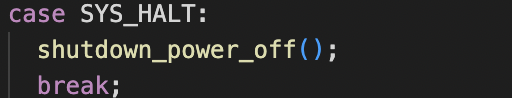
자동 생성된 설명

is\_user\_vaddr() 함수를 이용해 해당 value가 0이면 kernel값을 참조하고 있으므로

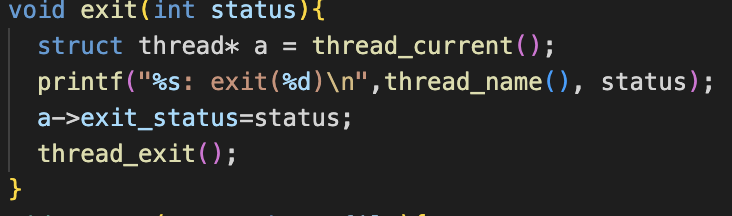
exit를 호출해 멈춰주었다.

3 System Calls

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



halt 시스템 콜은 따로 구현하지 않고 shutdown\_power\_off() 함수를 사용해 구현해 주었다.



exit 시스템콜은 현재 thread를 받아 theread의 상태를 저장하고 종료한다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

exec 시스템 콜은 process\_execute()함수를 이용한다. thread\_create() 의 첫번째 인자에는 thread의 이름이 들어가야 하므로 ‘echo x’를 parsing해 ‘echo’만 넣어줘야 하기 떄문에 parsing을 진행하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Wait 시스템 콜은 process\_wait()을 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread의 저장된 자식 list들을 순회 하면서 tid가 같으면 세마포어를 이용해 wait 해준다.

1. Additional System calls

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

텍스트이(가) 표시된 사진

자동 생성된 설명

Finonacci와 max\_of\_four\_int를 사용해 간단하게 구현해 주었다.

Max\_of\_four\_int는 인자가 4개 이기 때문에 새로운 syscall4를 구현해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**
* **텍스트이(가) 표시된 사진

  자동 생성된 설명**