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

담당 교수 : 박성용

조 / 조원 : 20191583 김태곤

개발 기간 : 2023.09.18 ~ 2023.10.08

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**
* Pintos는 기본적인 운영체제의 틀과 환경은 구성되어 있으나 세부적인 작동은 구현되어 있지 않다. System call이 호출되었을 때 Argument를 parsing하고, esp stack에 쌓아 passing하는 과정을 작성하고, halt, exit, exec, wait, read, write system call이 정상 작동하도록 구현한다. 또한 Fibonacci와 Max of four int 함수를 System call에 적절하게 추가하는 것이 이번 1차 프로젝트의 개발 목표이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

입력 받은 Argument를 공백을 기준으로 parsing한 후 esp stack에 규칙에 맞게 저장한다. 이를 통해 user code에서 handler까지 Parameter를 넘길 수 있다.

1. User Memory Access

주어진 주소의 유효성을 확인해야 한다. 총 stack size를 벗어났는지, 적절한 page를 가져왔는지, 잘못된 포인터를 참조하지 않았는지 확인한다.

1. System Calls

현재 대부분의 System call이 제대로 구현되어 있지 않다. 프로젝트1에서 제시된 requirements에 따라 halt, exit, exec, wait, read, write를 구현하여 System call이 정상 작동하도록 완성한다.

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

먼저 pintos의 run 명령어를 통해 실행시켜야 할 명령을 입력 받는다. 해당 명령은 file\_name으로 전달받고, file\_name을 공백을 기준으로 parsing하여 argv(사용자 선언)에 저장한다. 저장된 argv를 이용하여 esp stack에 argv 뒤에서부터 차례로 쌓고, world align을 통해 크기를 맞춘다. 이후 Null값을 한번 넣고 기존에 쌓았던 argv 값들의 주소 값을 차례로 쌓는다. 마지막으로 argc의 값을 쌓고, 0을 쌓아 esp를 구성한다.

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

Invalid memory access는 주로 유저모드에서 커널 주소에 접근하려고 시도할 때 발생한다. 유저 모드에서는 커널 영역의 주소에 직접 접근할 수 있는 권한이 없으며, 이러한 접근 시도는 보안 및 시스템의 안정성을 위해 방지되어야 한다.

* + Invalid memory access를 어떻게 막을 것인지 설명

Threads/vaddr.h에 있는 is\_user\_vaddr() 함수와 is\_kernel\_vaddr()함수를 이용해 Invalid memory access를 막는다. System call handler 과정에서 해당 system call의 함수를 수행하기 전, valid한지 is\_user\_vaddr()을 통해 미리 확인한다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

System call은 운영체제의 핵심 기능을 외부에서 사용할 수 있게 해주는 중요한 메커니즘이다. 운영체제는 컴퓨터 시스템을 관리하고 제어하는 역할을 하며, 이를 위해 하드웨어와 소프트웨어 간의 인터페이스 역할을 한다. 그러나 운영체제가 모든 기능을 외부에 노출시키면 보안과 안정성에 문제가 발생한다. 따라서 user mode와 kenel mode를 통해(dual-mode) critical한 접근을 막는다. 사용자는 운영체제에 권한이 필요한 작업을 수행하기 위해 system call을 호출한다. system call이 호출되면 커널 모드로 전환되며, 실행이 완료되면 다시 사용자 모드로 돌아간다. 이러한 restricted operations 문제를 해결하기 위해 system call이 필요하다.

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

halt() : 'devices/shutdown.h'에 선언된 shutdown\_power\_off()를 호출하여 Pintos를 종료한다.

exit() : 현재 사용자 프로그램(thread)을 종료하고 상태 정보(state)를 커널로 반환한다.

exec() : cmd line에서 지정된 명령을 실행하고, 주어진 argument를 전달 후 새로운 프로세스의 pid를 반환한다.

wait() : 자식 프로세스의 pid를 기다리고 자식 프로세스의 종료 상태를 검색한다. 만약 pid가 아직 활성상태라면, 해당 프로세스가 종료될 때까지 대기한다.

write() : fd가 1일 때 버퍼에서 fd로 열린 파일로부터 size만큼 바이트를 쓴다. 실제로 쓴 바이트 수를 반환한다.

read() : fd가 0인 경우 size만큼 바이트를 버퍼로 읽어온다. 실제로 읽어온 바이트 수를 반환한다.

fibonacci() : n번째 피보나치 숫자를 출력한다.

max\_of\_four\_int() : 입력 받은 4 정수 중 가장 큰 값을 출력한다.

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

User process에서 calls system call이 발생하면 lib/user/syscall.c에 있는 syscall이 수행된다. Syscall은 0x30 interrupt number로 정의되어 있으며, 해당 인터럽트 명렁어를 통해 커널 영역으로 전환을 요청하고, Interrupt Vector의 0x30 위치에 있는 syscall\_handler()를 호출한다. 커널 영역으로 전환을 요청할 때 system call number와 arguments 또한 함께 전달된다. Userprog/syscall.c 에 있는 syscall\_handler에서 넘겨받은 system call number에 따라 미리 정의된 함수들을 수행시킨다. System call 처리가 완료되면, syscall\_handler() 함수가 결과와 함께 ret을 통해 사용자 레벨로 복귀한다.

1. **추진 일정 및 개발 방법**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**
  1. **추진 일정**
* 09.18 ~ 09.22 : Pintos 프로젝트의 전반적인 코드 분석
* 09.23 ~ 09.24 : Argument parsing, passing 구현.
* 09.25 ~ 09.26 : Pintos 매뉴얼을 통한 system call 공부
* 09.27 ~ 10.02 : 항저우 아시안 게임 응원단(직관)으로 인한 개발 임시 중단
* 10.03 ~ 10.06 : system call과 user memory access 구현
* 10.07 : fibonacci와 max of four int 추가 구현
* 10.08 : 추가적인 점검 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)**
* Argument parsing & passing

process.c의 load함수에서 file\_name으로 넘겨받은 명령을 공백 기준으로 분리하여 argc를 구한다. 이후 argv를 argc만큼 할당하고, 공백 기준으로 해당 문자의 길이만큼 memcpy를 통해 저장한다. Parsing한 데이터를 2-B-Argument Passing에 설명대로 esp에 쌓는다.

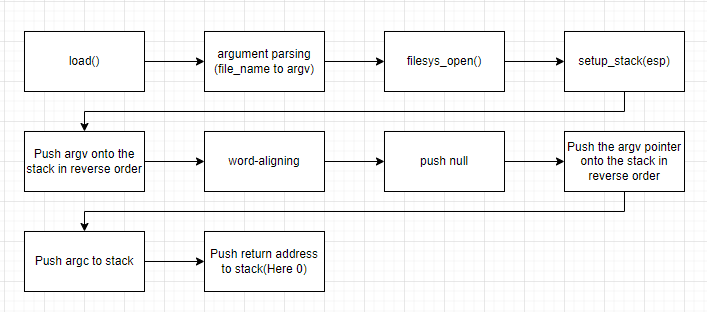
* System call & User Memory Access

Userprog/syscall.c에서 2-A-3에 설명대로 각 함수들을 정의한다. Syscall\_handler에서 esp를 통해 넘겨받은 syscall number를 switch문을 통해 구별하여 정의된 함수들을 수행시킨다. 함수 수행 전, is\_user\_vaddr을 통해 Memory Access가 valid한지 판단한다.

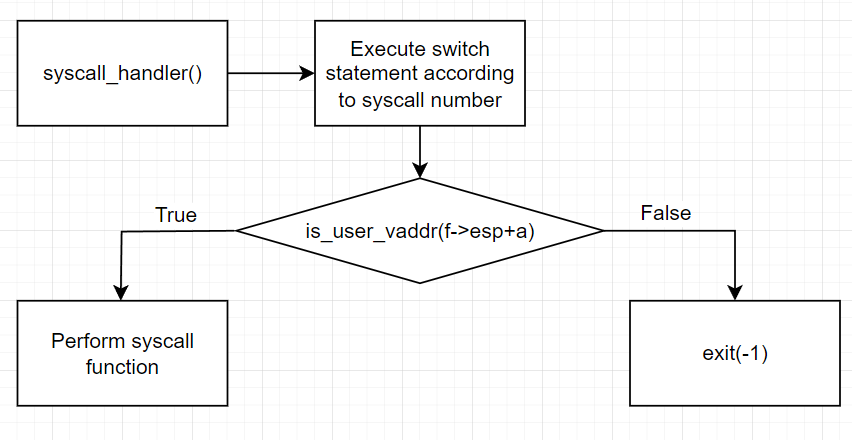
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

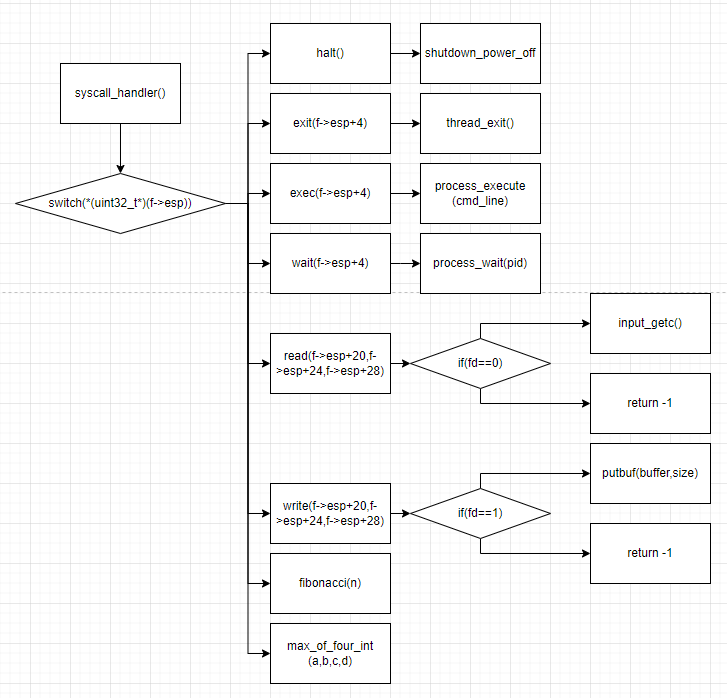
1. Argument Passing



1. User Memory Access

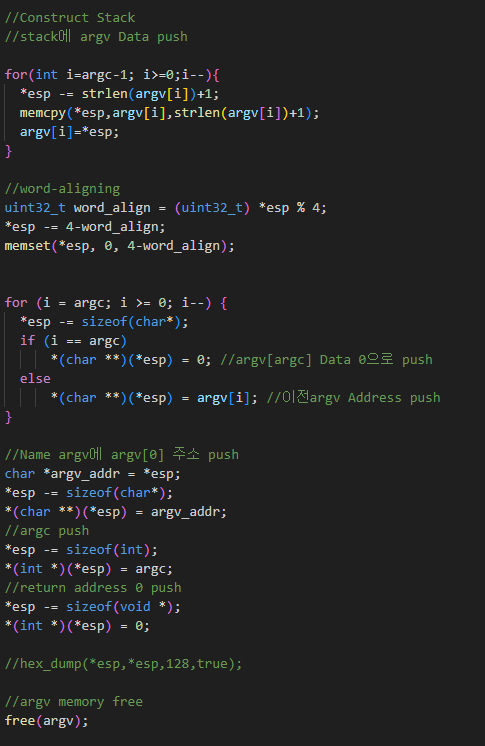
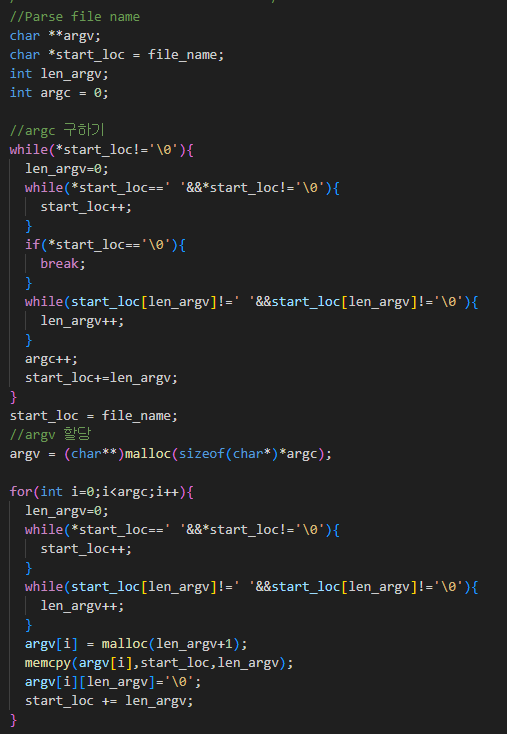


1. System Calls



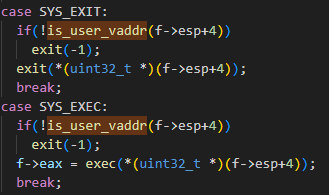
* 1. **제작 내용**
* **II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.**
* **구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명.**
* **개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.**

1. Argument Passing



위 과정을 Argument를 Parsing하고, Parsing된 argv를 esp에 쌓는 과정이다. Parsing 방법은 3-B의 설명과 동일하다. Passing과정은 먼저 strlen(argv[i])+1만큼 esp에서 빼주어 공간을 만들고 memcpy를 통해 argv를 뒤에서부터 삽입하였다. 이후 word-align 과정을 위해 4-esp%4의 결과를 esp에서 빼주었다. 생겨난 자리에는 0을 대입하였다. Argv[argc]의 위치에는 0을 대입하고, 이전 argv의 Address를 push해 주었다. 마지막으로 Name argv에 argv[0] 주소를 넣고, argc와 0을 push하여 esp stack을 통한 Argument Passing을 준비하였다.

1. User Memory Access



User memory Access부분은 userprog/syscall.c의 syscall\_handler 부분에서 각 함수를 실행시키기 전 is\_user\_vaddr을 통해 valid한지 파악하고, invalid한 경우 exit(-1)을 수행시켜주었다. 이를 통해 잘못된 메모리 접근을 예방하였다.

1. System Calls

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

userprog/syscall.c에서 각 System calls 함수들을 정의해 주었다. syscall\_handler 함수에서 매개변수로 받은 구조체 intr\_frame \*f를 통해 Argument Passing과정에서 구현한 esp를 넘겨 받아 각 변수와 Syscall number에 접근하였다. 가장 처음 f->esp의 값을 확인하여 해당 Syscall number를 확인하고, 이후 알맞은 함수를 esp에 있는 변수를 활용하여 호출하였다.

-halt() : halt 함수는 shutdown\_power\_off()를 이용하여 강제 종료를 구현하였다.

-exit() : exit 함수는 먼저 exit될 thtread의 이름(status)를 출력하고, thread\_current를 업데이트 한 후 thread\_exit()을 호출하였다. 이 과정에서 esp를 통해 넘겨받은 인자 f->esp+4를 활용하였다. 이 상태로 실행시키면 출력 결과에 file\_name : exit(0)가 뜨는데, 실제 필요한 정보는 file\_name 전체가 아닌 명령어 부분이다. 이를 해결하기 위해 userprog/process.c의 process\_excute에서 file\_name을 parsing하여 thread\_create로 넘겨주었다. 그 후 수행 결과를 확인해 보면 sc-bad 테스트 케이스에서 Page fault가 발생했었다. Userprog/exception.c에서 page\_fault 함수를 보면 page\_fault의 원인을 분석하는 코드가 존재한다. 이 부분에서 user가 커널 모드를 참조하려 하면 exit(-1)을 호출하여 오류를 수정하였다. userprog/process.c의 process\_wait 마지막 부분에 sema\_up, sema\_down과정이 추가되었는데, 이 부분은 관련된 wait을 설명할 때 추가 설명하겠다.

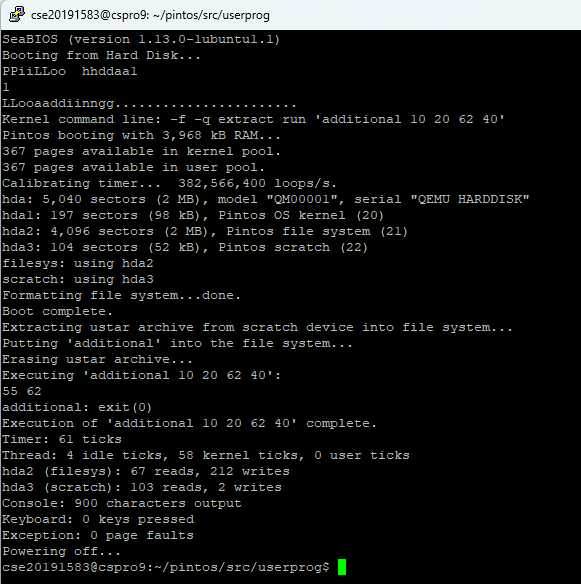
-exec() : exec함수는 process\_excute(cmd\_line)을 통해 입력 받은 명령을 수행하도록 구현하였다.

-wait() : wiat함수는 process\_wait(pid)를 통해 자식 프로세스의 pid를 기다리고 자식 프로세스의 종료 상태를 검색한다. 만약 pid가 아직 활성상태라면, 해당 프로세스가 종료될 때까지 대기한다. 이 과정에서 두개 이상의 멀티 thread가 사용되는 환경이므로 synchronization 과정이 필요하다. 이를 위해 thread 구조체에 semaphore를 위한 child\_semaphore와 memory\_semaphore를 추가하고, child list와 child\_element를 담는 child\_elem을 추가하였다. Process.c의 process\_wait 함수에서 child를 탐색하면서 child의 tid인 경우 sema\_down을 통해 child 가 끝날 때까지 기다린다. 이후 child가 process\_exit을 수행할 때 sema\_up을 통해 child가 기다리는 과정이 끝나고, parent 쪽에서 list\_remove 과정이 이루어진다. 잘 작성된 코드 같지만 이렇게만 작성할 경우 제대로 된 작동이 이루어지지 않는다. child process가 sema\_up을 하고 죽었을 때 list\_remove에서 필요한 child의 메모리가 안 남아 있을 수 있다. 때문에 memory\_semaphore를 활용하여 list\_remove에 필요한 메모리를 남기는 작업을 추가하였다. child에서 sema\_up을 통해 종료를 알리고, memory 보존을 위해 sema\_down(memory\_semaphore)를 해준다. 이후 parent에서는 child\_semaphore가 정상으로 돌아왔으니, 그 이후 list\_remove 과정이 수행되고, sema\_up(memory\_semaphore)를 통해 정상적으로 child process가 처리된다.

-read() : read함수는 fd가 0일 때 input\_getc() 함수를 통해 읽어오도록 구현하였다. 이때 필요한 fd와 buffer, size가 16만큼 더 밀려 있어(hex\_dump를 통해 확인 가능) 기존 다른 함수들과는 달리 f->esp+20부터 인자로 넘겨주었다.

-write() : write함수는 fd가 1일 때 putbuf(buffer,size)를 통해 size만큼 buffer에 쓰고, 실제 쓴 byte 수(size)를 반환하였다. write함수 또한 16만큼 밀려 있어 f->esp+20부터 인자로 넘겨주었다.

1. Additional System calls

* **새로운 시스템 콜(fibonacci, max\_of\_four\_int)을 구현하기 위해 수정하거나 작성한 코드에 대해 서술**
* 먼저 추가 system calls을 구현하기 위해 lib/user/syscall.h 두 system call API를 추가하였다. 선언한 함수를 바탕으로 lib/user/syscall.c에서 각각 syscall1(SYS\_FIBO,n)과 syscall4(SYS\_MOFI,a,b,c,d)를 호출하는 함수를 정의하였다. SYS\_FIBO와 SYS\_MOFI는 lib/syscall-nr.h에 enum으로 정의된 syscall number에 추가하였다. 이후 userprog/syscall.h에 두 함수를 선언하고, userprog/syscall.c에서 각 함수에 대해 정의하였다. Fibonacci 함수는 n이 0일때는 0, 1일때는 1을 반환하고, 2 이상부터 반복문을 통해 이전 값을 더해가면서 구한 후 반환하였다. Max\_of\_four\_int 함수는 처음 a값을 max로 지정하고, 이후 b,c,d 값을 차례로 비교하며 더 큰 값이 들어오면 max값을 업데이트 하고 반환해 주었다. 최종적으로 examples 폴더에 additional.c 파일을 만들어 입력 받은 argv를 이용해 두 함수를 출력하는 파일을 만들었다. 이때 argv는 string으로 입력 받으므로, str\_to\_int 함수를 선언하여 숫자로 바꾸는 과정을 거쳐 두 함수의 인자로 넘겨주었다.
  1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**
* ****