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

담당 교수 : 김영재

조 / 조원 : 조린

개발 기간 : 3주

1. **개발 목표**

pintos상에서 프로그램이 정상적으로 실행되도록 argument를 user stack에 쌓고, 프로그램에서 kernel에 system call을 하면 kernel api를 통해 적절히 수행함. 그 과정에서 user stack에 접근하는 것을 보장해주도록 함.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Argument Passing

프로그램 수행 시 전달한 argument를 x86 convention에 따라 user stack에 저장한다.

1. User Memory Access

System call에 사용된 argument를 확인하는 과정에서 user stack에 접근하고 있는지 확인하고, kernel stack또는 비정상적인 주소라면 종료한다.

1. System Calls

User stack에 저장되어있는 argument를 가지고 kernel의 api를 사용한다.

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

file\_name을 load하는 과정에서 file\_name을 공백단위로 분리 해 마지막 단어부터 user stack에 쌓는다. null을 통해 변수가 끝임을 알리고, 각 argument가 stack상에서 저장된 주소를 한 번더 쌓는다. 그 뒤에 argv의 시작 주소와 argc를 쌓고, return address(default = 0)를 쌓는다.

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

User program이 사용하는 메모리의 크기는 4GB고, 0xC0000000에서 시작해 주소가 낮아지는 방향으로 stack이 성장하게 된다. kernel api를 사용하기 위해 esp를 전달하는 과정에서 esp가 user stack을 가리키지 않는 경우(kernel stack이거나 page fault 발생 시) 즉시 프로그램을 종료함.

* + Invalid memory access를 어떻게 막을 것인지 설명

Src/threads/vaddr.h에 저장되어있는 is\_user\_addr(esp)를 통해 user stack address가 아니라면 종료함. 만약 esp가 0아래의 주소를 가리킨다면 page fault가 발생하는데 이때 page fault를 조정해 에러메세지 출력이 아닌 프로그램 종료로 변경함.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

User program의 유연성을 높이기 위해 file system에 접근하거나, 다른 process와 상호작용할 때 제한적으로 수행하기 위해 kernel상에서 수행하도록 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() : shutdown\_power\_off()를 통해 수행중인 process를 중지시킴.

Exit() : process가 종료될 때마다 종료 status를 parent process에게 전달하고 화면상에 출력한 뒤 사용되던 자원을 회수하고 종료함.

Exec() : cmd\_line을 parsing한 뒤, argv[0]에 해당하는 program이 존재한다면 수행하고, 아니면 -1을 반환함.

Wait() : process에서 생성된 child process가 모두 종료될 때까지 기다리도록 대기하다 child process가 종료되면 child process의 exit status를 저장함.

Read() : stdin(0)으로 들어온 input을 buffer에 최대 size만큼 저장함.

Write() : stdout(1)로 buffer를 최대 size만큼 출력함.

Fibonacci() : n번째의 피보나치 수를 반환함.

Max\_of\_four\_int() : 4개의 argument중 가장 큰 수를 반환함.

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

syscall을 통해 수행할 명령(ex : SYS\_WAIT)과 argument를 stack에 저장한 뒤 intr\_handler로 이동함. intr\_handler에서 syscall\_handler를 호출하고, kernel에서 esp를 통해 명령과 명령에 맞는 argument를 받아와 명령을 수행함. 반환값이 있을 경우 eax register에 값을 저장한 후 user program으로 돌아옴.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

9/11 ~ 13 : argument passing: stack에 변수를 저장

9/14 ~ 15 : user memory access : esp가 user memory를 가리키고 있는지 확인 후 esp또는 argv의 주소가 kernel memory일 경우 발생하는 page fault를 exit하도록 구현

9/23 ~ 25 : system calls(halt, exit, exec, write(stdout))이 정상적으로 수행되도록 구현

9/26 : multi-recurse에서 발생하는 오류 수정(exec시에도 parsing을 통해 파일명을 확인하도록 함)

9/27 : additional 과제를 수행하는 fibonacci(), max\_of\_four\_int() 작성

9/28 : read(stdin)이 정상적으로 수행되도록 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 passing : src/userprog/process.c에서 option과 argument를 분리해 stack에 쌓아야 하기 때문에 strtok\_r()을 사용해 공백 단위로 단어를 분리해내는 parsing(char \*src, char \*\*argv)함수 구현.

생성된 esp에 argv를 저장하기 위한 struct\_stack(void \*\*esp, int argc, char \*\*argv)구현.

User memory access : src/userprog/syscall.c에서 user virtual memory에 접근하는지 is\_user\_vaddr()을 추가함.

비정상적인 주소일 경우 발생하는 src/userprog/exception.c에서 user virtual address가 아닌 상태에서 발생하는 page fault에는 프로그램을 종료하도록 변경.

System call : src/userprog/syscall.c에서 syscall\_handler()가 esp에 저장되어있는 명령에 따라 변수를 esp에서 가져와 적절한 함수를 수행하도록 함. halt(), exit(), exec(), wait(), read(), write()함수 구현.

Src/threads/thread.h에서 struct thread에 child process를 기다릴 수 있도록 struct list child\_list, struct list\_elem child\_elem, struct semaphore child\_done을 추가.

child process의 exit\_status를 읽는 동안 child process가 종료되지 않도록 하는 struct semaphore removal\_complete 추가.

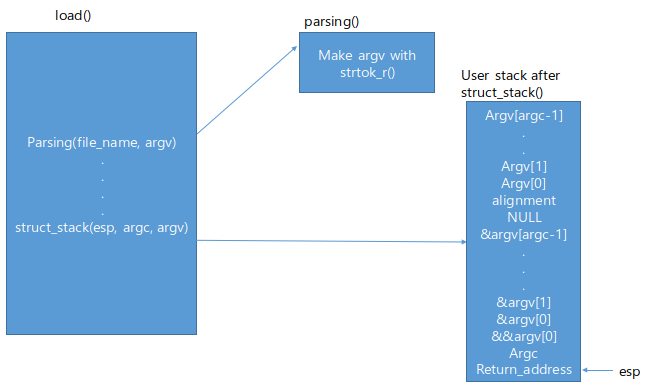
child process의 exit status를 저장하는 int exit\_status추가.

Src/userprog/process.c에서 exec()시에 process\_execute()에서 argument를 분리해 file\_name만을 변수로 넘겨주도록 코드 추가.

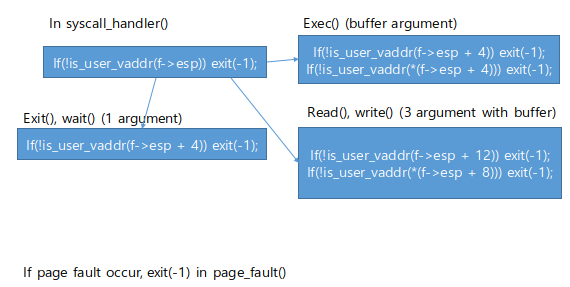
변수로 들어온 tid를 확인하고, semaphore를 통해 child가 끝나는 것을 기다린 뒤, removal\_complete를 통해 child process가 종료되도록 구현.

Exit()의 경우 parent process에 자신이 종료되었음을 알리고, parent process가 자신의 정보를 다 지우는 것을 기다렸다가 removal\_complete를 통해 끝난 것을 확인 받고 종료되도록 구현.

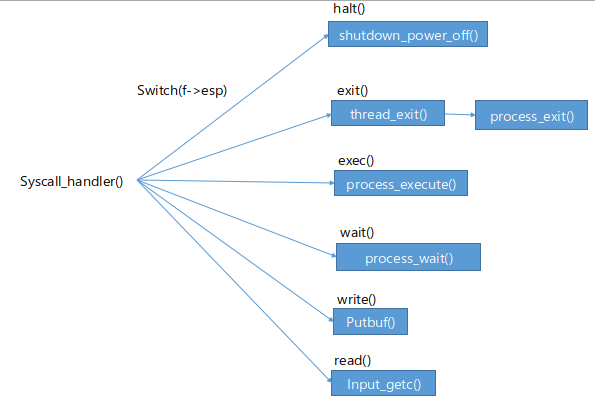
1. **연구 결과**
   1. **Flow Chart**
2. Argument Passing



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

input으로 들어온 file\_name을 공백단위로 나눠 file name과 option, argument를 구한 뒤 argv에 저장하고 argc를 반환하기 위해 strtok\_r()을 사용한 parsing()을 구현함.

Argc, argv를 esp를 통해 user stack에 쌓아야 하는데 80x86 convention에 따라 argv의 크기만큼 esp를 감소시키고, esp의 위치에 argv를 쌓는 작업을 반복문을 통해 수행하고, stack bottom에서 argv의 원소의 크기만큼 증가시킨 위치를 stack에 push한 뒤 esp의 위치를 한 번 더 저장한 뒤, argc를 push하고, return address(default = 0)을 push한 뒤 true를 반환하는 struct\_stack()을 구현.

1. User Memory Access

struct\_stack()에서 argv가 저장될 위치가 user virtual address인지 확인하기 위해 is\_user\_vaddr()을 struct\_stack()에 추가함.

Syscall\_handler()에서 최초 f->esp가 user virtual address인지 확인하고, 각 명령에 맞는 변수들의 위치가 user virtual address인지 확인할 수 있도록 is\_user\_vaddr()을 변수들이 있는 위치에 대해 매 번 실시함.

kernel이 system call에 대한 코드를 수행하다 page fault가 발생했을 때 수행되는 page\_fault()에서 kernel이 발생시킨 fault라면 에러메세지 출력이 아닌 프로그램을 바로 종료하도록 코드 추가.

1. System Calls

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

User program이 system call을 하는 과정에서 esp에 user가 요청한 함수가 저장되어있고, switch를 통해 적절한 함수를 수행한다.

Halt() : 명세서에 나와있는 대로 shutdown\_power\_off()를 수행함으로써 프로그램이 중지될 수 있도록 한다.

Exec() : process\_execute()에 argument를 변수로 전달하며 호출한다.

* Multi recurse test의 경우 exec()이 비정상적으로 종료되었는데, argument까지 포함된 전체 file name을 load하려다가 생긴 문제였다. 따라서 exec()내에도 parsing을 통해 file name을 추출하는 과정을 거쳤다.

Exit() : thread\_exit()을 통해 현재 프로세스의 자원을 회수하고, 종료한다.

- parent process가 child process를 기다리지 않고 즉시 종료되면서 child process가 제대로 수행되지 않는 문제가 있었다. parent process는 semaphore를 통해 child process가 종료되는 것을 기다리고, child process는 종료시 parent process에게 알린 뒤 마찬가지로 semaphore를 통해 정보를 제거하는 것을 기다린다.

Wait() : 변수로 들어온 tid를 자신의 child\_list에서 찾고, semaphore를 통해 child process가 알릴 때 까지 기다린다. child process를 저장하고 있던 list를 제거한 뒤 child process에게 제거가 완료되었음을 알리고 child process의 exit status를 반환한다.

Read() : pintos 내장 함수인 input\_getc()를 이용해 최대 size만큼을 buffer에 저장한다.

Write() : pintos 내장 함수인 putbuf()를 이용해 최대 size만큼을 buffer에서 읽어와 화면에 출력한다.

1. Additional System calls

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

Src/lib/syscall-nr.h에서 SYS\_FIBO와 SYS\_MAXF를 추가해 새로운 system call에 대해 kernel로 권한을 넘길 수 있도록 함.

Lib/user/syscall.c에서 fibonacci()와 max\_of\_four\_int()가 변수의 개수에 맞는 syscall을 하도록 연결하고, max\_of\_four\_int()의 경우 변수가 4개이므로 syscall4()를 새로 추가함.

Userprog/syscall.c에서 syscall\_handler()에 SYS\_FIBO와 SYS\_MAXF가 왔을 때 맞는 함수를 연결지어 줄 수 있도록 switch 문을 추가함.

Userprog/syscall.h와 userprog/syscall.c에서 fibonacci와 max\_of\_four\_int를 수행하는 함수 fibonacci(int n)와 max\_of\_four\_int(int a, int b, int c, int d)를 추가함.

Fibonacci(int n) : int ret에 i번째 fibonacci 수를 저장하고, cur에는 i-1, before에는 i-2번째 fibonacci수를 저장하도록 반복문을 수행해 결과값(ret)을 반환함.

Max\_of\_four\_int(int a, int b, int c, int d) : 처음 값을 a로 설정한 뒤 변수를 비교하며 더 큰 값을 결괏값으로 설정하고, 4개의 변수를 모두 확인한 뒤 결과값(ret)을 반환함.

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

