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

담당 교수 : 김영재

조 / 조원 : 임나현

개발 기간 : 2024.9.17-2024.9.30

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

Pintos 상에서 shell과 동작이 유사하게 사용자에게 입력 받은 명령어를 처리할 수 있게 한다. 우선 입력받은 명령어가 stack에 제대로 전달이 되도록 parsing하는 과정이 필요하고, 명령어들이 호출하는 system call이 작동하도록 구현한다. 추가적으로 새로운 system call (Fibonacci, max\_of\_four\_int)를 구현해본다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

argument로 주어지는 file\_name을 stack에 공백 단위로 저장하고 관리하기 위해서는 먼저 parsing 과정이 필요하다. Argument을 공백을 기준으로 이차원배열에 동적 할당하여 저장한다. 그 후, stack에 순서대로 passing하여 저장한다. hex\_dump를 사용하면, stack에 제대로 저장이 되어 있는지 확인할 수 있다.

1. User Memory Access

Memory에는 kernel memory와 user memory로 구분되어 있다. System call로 넘어온 주소가 user memory 내의 위치를 가리키고 있어야 하는데 그렇지 않은 경우 kernel 영역을 침범할 가능성이 있다. 만약, kernel 영역을 침범했다면 exit system call을 통해 종료 시켜야 한다.

1. System Calls

System call에는 총 6개의 system calls이 구현이 되어야 한다. 이는 halt, exit, exec, wait, read, write system call이다. 추가적으로 shell에는 없는 새로운 system call을 구현해야 하는데, 이는 Fibonacci와 max\_of\_four\_int이다.

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

process.c 파일 내에 load 함수에서 filesys\_open을 하게 되어 있다. Open을 실행하기 전, 현재는 입력 받은 문자열 전체가 저장되어 있기 때문에 parsing 과정이 필요하다. parsing은 공백을 기준으로 분리하여 이차원 배열에 동적 할당하여 저장한다. 저장한 argument들을 stack에 쌓아 주는데 80X86 convention에 맞춰야 한다. \*esp = PHYS\_BASE로 설정하고, argument의 길이에 맞춰 argument들을 stack에 저장한다. 저장이 완료되면, 글자수에 따라 word alignment가 안 맞는 경우가 생기므로, 추가적으로 4 word alignment를 맞춰 줘야 한다.

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

Pintos에서 설정되어 있는 valid 메모리 범위를 넘어선 주소를 접근하려고 했을 때 page fault가 발생한다. 이는 user memory 영역이 아닌 PHYS\_BASE 아래에 있는 kernel memory 영역을 접근하려고 시도했기 때문이다. 이러한 경우 invalid memory access에 해당하여 예외처리를 통해 exit하게 해야 한다.

* + Invalid memory access를 어떻게 막을 것인지 설명

threads/vaddr.h에 정의되어 있는 is\_user\_vaddr 함수와 userprog/pagedir.c의 pagedir\_get\_page 함수를 이용할 수 있다. 두 함수 모두 현재 주소가 알맞은 user memory 위치를 가리키고 있는지 확인할 수 있다. 이 함수를 통해 kernel memory를 접근하려고 하거나, 주소 자체가 NULL 값이라면 invalid memory access이기 때문에 exit한다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

System call은 user mode와 kernel mode 간의 인터페이스로, 어떤 프로그램의 요청 (ex. File read)에 따라 커널에 접근할 수 있도록 한다. 사용자가 직접적으로 하드웨어 장치를 제어하게 되면 큰 문제가 생길 수 있으므로, 하드웨어에 대한 접근이 필요할 때, 사용자는 system call을 통해 kernel에게 처리를 요청할 수 있다.

* + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명

Halt: shutdown\_power\_off 함수를 이용해서 pintos를 종료한다.  
Exit: thread 이름과 exit status를 출력하고 process를 종료한다. status number는 커널에 보고한다.  
Exec: 인자에 해당하는 process를 process\_execute함수를 통해 실행한다.  
Wait: child process가 종료되기를 parent process는 기다리고, 종료된 후에 child process의 exit status값을 받는다.  
Read: STDIN file descripter로 입력 받은 문자를 buffer에 저장한다.  
Write: STDOUT file descripter로 인자로 받은 buffer를 출력한다.  
FIBONACCI: 인자로 받은 정수 값에 해당하는 피보나치 값을 반환한다.  
MAX\_OF\_FOUR\_INT: 인자로 받은 4개의 정수 값 중에서 가장 큰 값을 반환한다.

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

User level에서 system call 호출한다. 이때 user stack에 system call number와 argument가 저장된다. 이후, kernel mode로 전환되어 system call number를 확인하여 해당되는 system call 함수를 실행시키고 user level에서 요청한 동작을 한다. 실행된 후, kernel mode에서는 return값을 user mode에 전달하기 위해 stack에 저장하고 다시 user mode로 프로세서의 제어를 돌려준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

2024.09.17 - 2024.09.22: argument parsing과 stack 저장 구현

2024.09.23 – 2024.09.29: user memory access, system call 공부 및 구현

2024.09.30: additional system call 구현

2024.10.02: 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)**

1. argument passing

-userprog/process.c/load function

load 함수에 filesys\_open 전에 공백을 기준으로 parsing하는 과정이 필요하다. argument들을 나눠서 동적 할당을 통해 배열에 저장한다. filesys\_open의 인자에는 분리된 첫 번째 argument를 넘겨준다. 이후 userstack을 동적할당으로 새로 생성하고, 아까 공백을 기준으로 분리한 argument들을 stack에 쌓는다. \*esp = PHYS\_BASE로 설정하고, argument들의 길이에 맞게 80x86 convention을 맞춰 쌓아 준다. argument들의 길이와 상관없이 4 word alignment를 만족해야 하기 때문에 추가 변수를 사용하여 맞춰준다. 총 stack에 들어가는 것은 각각의 argument, argument 주소, arguments의 개수, return address가 있다.

1. User memory access

-userprog/syscall.c, userprog/exception.c, threads/vaddr.c

vaddr.c에 정의되어 있는 is\_user\_vaddr 함수와 is\_kernel\_vaddr 함수를 이용하면 메모리 주소가 user memory 영역을 가리키고 있는지 확인할 수 있다. exception.c에서 page\_fault에 의해 kernel 영역에 접근하고 있는지 확인할 수 있다. 이를 적용하여 syscall.c에 address가 유효한지 확인하는 함수를 구현하여 만약, kernel 영역을 접근한다면 비정상 exit한다.

1. system call

-userprog/syscall.c/syscall\_handler function, userprog/process.c

syscall\_handler 함수에서 switch문을 활용하여 halt, exit, exec, wait, read, write syscall마다의 작업을 작성한다. 실제로 작업을 수행하기 전에, 인자로 받은 주소가 유효한지 확인하는 함수가 필요하다. 작업을 완료한 후에 반환 값을 eax register에 저장한다. wait의 경우, parent process가 child process를 기다려야 하기 때문에 userprog/process.c에 semaphore를 추가로 작성한다.

1. additional system call

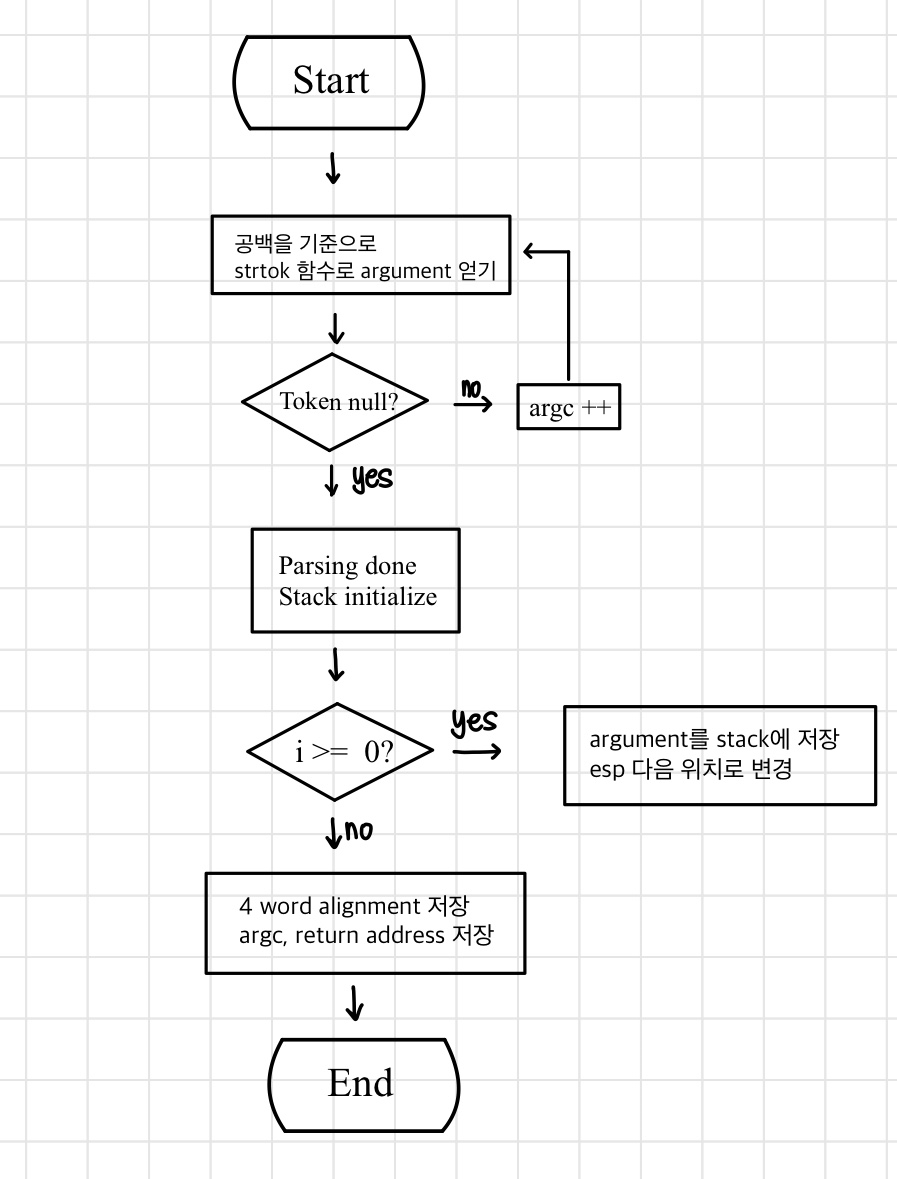
-lib/sycall-nr.h, lib/user/syscall.c, examples/additional.c

lib/syscall-nr.h에 FIBONACCI, MAX\_OF\_FOUR\_INT에 해당하는 syscall을 새롭게 추가한다. MAX\_OF\_FOUR\_INT는 인자가 4개가 필요하기 때문에 syscall4를 추가해야 한다. 추가로 pintos에는 피보나치와 최대값을 찾는 example이 없기 때문에 추가해야 한다.

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing



1. User Memory Access

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

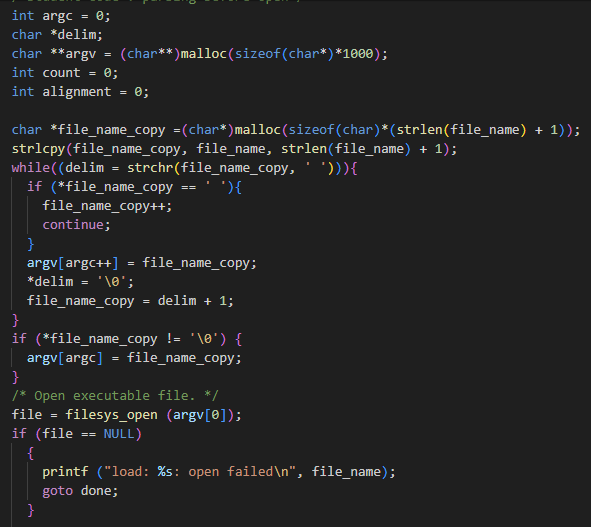
1. System Calls

도표, 라인, 평면도, 기술 도면이(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



userprog/process.h/load 함수 내에서 argument를 parsing하는 부분이다. 우선 argument들이 여러 개 포함되어 있는 문장을 file\_name\_copy에 복사한다. ‘ ‘을 기준으로 동적할당한 argv 이차원 배열에 저장하고 argc 개수를 +1 증가한다. Filesys\_open의 인자로는 문장 전체 대신 첫 번째 argument를 넘긴다.

**텍스트, 스크린샷이(가) 표시된 사진

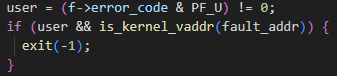
자동 생성된 설명**

userprog/process.h/load 함수 내에서 user stack을 만드는 과정이다. esp 포인터는 PHYS\_BASE로 초기화하고 argument의 주소를 저장할 동적할당 mystack 이차원 배열을 만든다. 그 후, 각각의 argument의 길이만큼 esp 포인터를 조정해가면서 argument 전부를 stack에 쌓는다. 쌓고 난 뒤에, 4 word alignment를 만족할 수 있도록 하는 과정이 필요하다 마지막으로, argv 주소, argument 개수, return address를 stack에 쌓는다.

1. User Memory Access

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명



텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

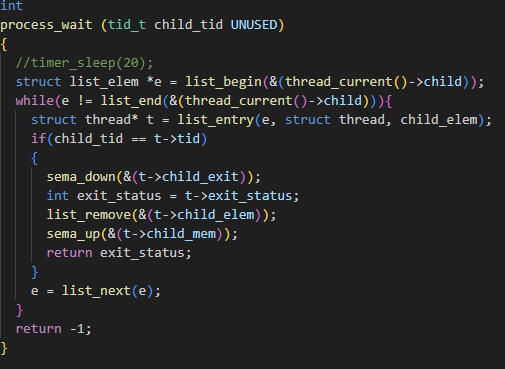
userprog/syscall.c/check\_addr 함수를 새로 정의하여 만약 인자로 주어진 address가 NULL값이거나 kernel 영역에 접근하는 주소이면 exit시킨다. Userprog/excetion.c에는 접속자가 user인데 kernel 영역에 접근하려고 한다면 exit시키는 코드를 추가한다.

1. System Calls

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

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**



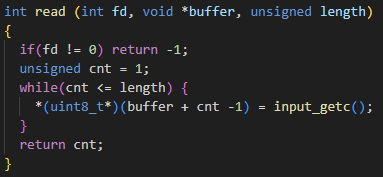
Userprog/syscall.c/syscall\_handler 함수 내에 switch 문을 활용하여 각각의 system call을 처리하였다. SYS\_HALT, SYS\_EXIT, SYS\_EXEC, SYS\_WAIT, SYS\_READ, SYS\_WRITE를 구현하였고, 각 system call은 실행 전에 인자로 받은 주소가 유효한지 먼저 체크한다. 그 후, 각각의 동작을 실행한다. 예를 들어, wait의 경우 userprog/process.c/process\_wait 함수를 실행시킨다.

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

userprog/syscall.c/exit 함수이다. Thread의 이름, status를 출력하고, 현재 thread에 exit\_status에 status를 저장하고 thread\_exit함수를 실행한다. Exit\_status는 threads/thread.h thread 구조체에 추가한다.

****

userprog/syscall.c/read 함수이다. length만큼의 길이를 buffer에서 한 글자씩 input\_getc()를 사용하여 읽고 읽은 개수를 return한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

userprog/syscall.c/write 함수이다. Buffer에 length만큼을 write한다.

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

****

userprog/process.h/process\_wait 함수와 process\_exit함수이다. Semaphore를 사용하여 lock하는 과정이 필요한데, child process가 종료되기 전까지 parent process는 종료될 수 없도록 해야 한다. 만약, parent process에서 list\_remove할 자식 메모리가 남아있지 않는 경우, 코드가 정상적으로 작동이 되지 않는다. 따라서 parent가 list\_remove하기 전까지 child process는 child\_memory에 한해서 lock을 하는 과정이 필요하다.

1. Additional System calls

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

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명





텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

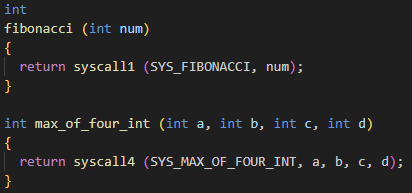
자동 생성된 설명

userprog/syscall.c/syscall\_handler 함수의 switch 문에 fibonacci와 max\_of\_four\_int를 추가한다. Lib/syscall-nr.h와 userprog/syscall.h에도 두 개의 함수를 추가한다.

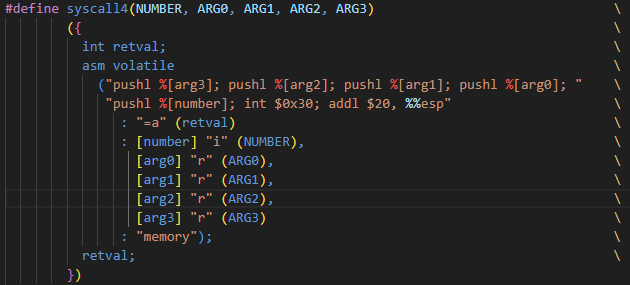
fibonacci 함수는 0일 때는 0을 반환, 1일 때는 1을 반환하고, 나머지의 경우에는 재귀함수를 사용하여 피보나치수열을 계산한다.

max\_of\_four\_int함수는 네 개의 정수 중에서 가장 큰 정수를 반환한다. 두 개씩 비교하여 큰 수를 찾고, 큰 수 두 개를 비교하여 최종적으로 가장 큰 정수를 반환하도록 작성하였다.





lib/user/syscall.c와 lib/user/sycall.h에 fibonacci와 max\_of\_four\_int 함수를 추가한다. fibonacci는 인자가 1개 필요하므로 syscall1을, max\_of\_four\_int는 인자가 4개 필요하므로 syscall4를 정의해야 한다.



lib/user/syscall.c에 syscall4함수를 정의한 모습이다 argument 4개를 push하여 사용하는 system call을 처리한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Examples/additional.c 파일을 만들어, 주어진 4개의 인자를 받아 Fibonacci 실행 값과 max\_of\_four\_int 함수의 실행 값을 출력하는 example를 만든다.

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

**텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명**

10 20 62 40의 4개의 인자를 주어줬을 때, Fibonacci 값은 55, max\_of\_four\_int 값은 62로 출력된다는 것을 확인할 수 있다.