**-**

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

담당 교수 :

조 / 조원 :

개발 기간 :

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

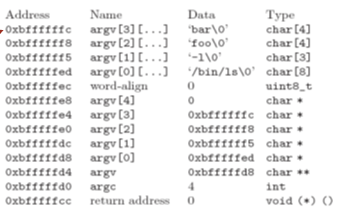
제공된 핀토스에는 system call, system call handler, argument passing, user stack이 구현되어 있지 않다. 그래서 command를 실행할 수 없다. 따라서 위에 나열한 것들을 구현하여 핀토스가 정상 실행하게끔 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

입력한 command의 argument들과, argument들이 담긴 주소들, 그리고 argc가 80x86 calling convention에 따라 user stack에 저장된다.



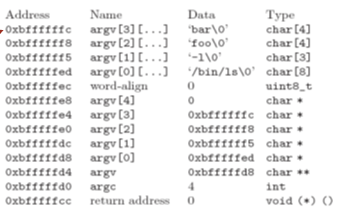
1. User Memory Access

invalid한 pointer들은 핀토스에서 거부한다. invalid한 pointer은 대체적으로 다음과 같다. Unmapped virtual memory거나, user address가 아닌 kernel address를 가리킨다.

1. System Calls

exec(), exit(), write(), read()와 같은 system call을 실행할 수 있다.

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



80x86 calling convention과 같이 위의 순서대로 argument를 스택에 쌓는다.

argument들은 file\_name에서 파싱한다.

방법은 다음과 같다. void \*\*esp의 주소를 PHYS\_BASE로 한다.

void \*\*esp는 앞으로 저장할 argument들의 위치를 지정하는 포인터이다.

argument의 크기에 맞게 void \*\*esp의 주소를 decrement해준 후,

library에서 주어진 함수(memcpy)를 이용하여 해당 주소에 argument를 저장한다.

word-align의 크기는 저장한 정보들의 크기를 고려하여 적절하게 계산해준다.

이후엔 argument들이 저장된 주소와 argv, return address를 저장한다.

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

1. access한 memory가 unmapped virtual memory인가?
2. 해당 pointer가 user space가 아닌 kernel space를 가리키는가?
3. 해당 pointer가 NULL인가?
   * Invalid memory access를 어떻게 막을 것인지 설명

언급하는 함수들은 userprog/pagedir.c, threads/vaddr.h에 위치한다.

1. is\_user\_vaddr(address), is\_kernel\_vaddr(address) 이용
2. pagedir\_get\_page( ) 이용

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

OS는 user program이 kernel memory를 access하는 것을 막는다. kernel memory는 핵심 functionalities를 갖고 있다. 그렇다면 user program은 kernel의 functionality를 어떻게 사용하는가? 이를 위해 os에서 system call을 제공한다.

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

halt: pintos를 강제종료한다.

exit: 현재 user program을 종료한다. kernel에 status를 return한다.

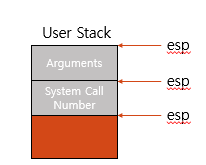
exec: child process를 만든다.

wait: child process가 끝날 때까지 기다린다. child process가 종료되었을 때 exit status를 받아온다.

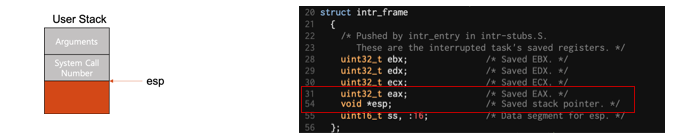
write: console에 buffer을 출력한다.

read: console에 buffer을 읽는다.

* + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명
    1. system call number와 추가적인 argument들이 caller의 stack으로 push된다.

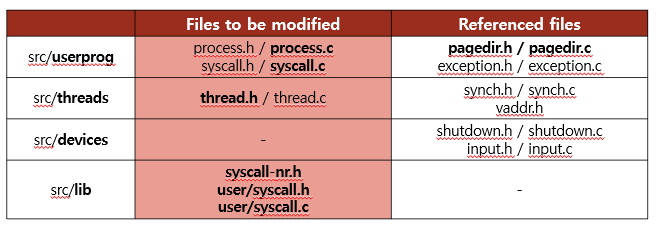


* + 1. 중간 작업(INT &030… 등)으로 syscall\_handler()이 control을 갖는다. 그리고 struct intr\_frame(threads/interrupt.h)의 member인 ‘esp’로 stack에 접근할 수 있다. system call api를 호출한 후 return 값은 strucgt intr\_frame의 member 인 ‘eax’에 저장한다.



1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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



process.c: process\_wait()에서 child thread가 exit되어 status를 return할 때까지 wait할 수 있도록 구현/process\_exit()에서 process가 끝날 때 semaphore signal작업을 통해 process\_wait()에서 sleep하고 있는 parant thread를 깨울 수 있는 작업 구현/load()에서 parameter parsing하여 user stack을 쌓을 수 있게끔 작업 구현

syscall.c: ../lib/syscall-nr.h에 주어진 system call number에 맞게 각 system call이 실행할 수 있도록 구현

thread.h/thread.c: process\_wait, process\_exit 등에서 쓰일 semaphore 선언

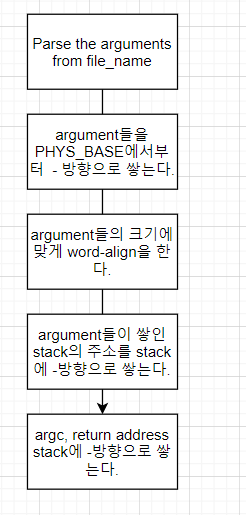
lib/syscall-nr.h: additional system call의 system call number 구현

lib/user/syscall.c: additional system call의 함수 구현

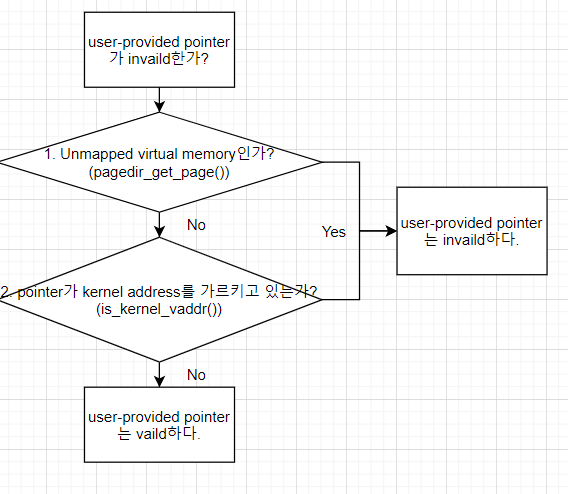
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

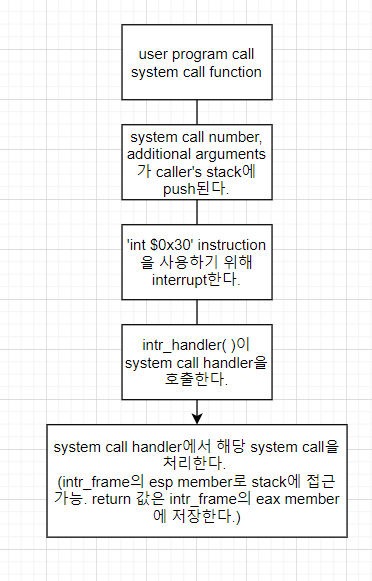


1. User Memory Access



*(가르키고 -> 가리키고)*

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

이 과정은 userprog/process.c의 함수 load에 진행된다. pintos에 내장한 strtok\_r 함수(strtok\_r(file\_name, delim, &save\_ptr): string file\_name에서 delim(ex: “ “)까지의 string을 떼어내고, 해당 string의 주소를 return)를 이용하여 parse한다. parse한 argument의 크기만큼 esp의 주소(esp의 주소는 setup\_stack(esp) 함수로 인해 PHYS\_BASE로 초기화되어 있다.)를 빼준 후, pintos에 내장된 memcpy(parameter: 저장하고자 하는 위치, 저장하고자 하는 내용, 저장하고자 하는 내용의 크기)[저장하고자 하는 내용을 저장하고자 하는 위치에 저장한다.] 을 이용해, stack에 저장한다. 이 때 argument의 크기를 모두 count한다. 그리고 word align해주기 위해 0을 채워야 하는 개수를 계산하여 이만큼 또 esp의 주소를 빼주고, 0을 저장한다. 이후엔 argument의 주소들과 return address, argc를 stack에 쌓는다.

1. User Memory Access

해당 주소가 unmapped virtual memory인지 체크하고 싶은 경우, call page\_dir\_getpage(주소), 해당 주소가 kernel 영역에 있는지 체크하고 싶은 경우 call Is\_kernel\_vaddr(주소)를 실행한다.

1. System Calls

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

System Calls은 pintos/src/userprog/syscall.c에 구현이 되어 있다.

System Call에 쓰이는 parameter들은 user stack에 저장이 되어 온다.

System call의 number는 (int\*)f->esp에, 첫번째 parameter의 주소는 ((int\*)f->esp + 1)에,

두번째 parameter의 주소는 ((int\*)f->esp + 2)에, n번째 parameter의 주소는 ((int\*)f->esp + n)

에 있다. (편의상 모든 paramter의 자료형을 int로 통일했다. 상황에 따라 parameter의 자료형

은 물론 달라진다. 편의상 이후에 stack에서 paramter으로의 접근 방법은 생략하겠다.)

-HALT: call shutfown\_power\_off()

-EXIT: 현재 돌아가고 있는 thread의 이름을 tokenize하여 가져온 후, 이름과 exit status를 출력한다. 그 후 thread\_current()->exit\_status를 status로 업데이트해준 후, thread\_exit()을 call하여 최종적으로 EXIT한다.

-WAIT: system call의 parameter로 child thread의 id가 온다. 이것을 wait해주기 위해 process.c의 process\_wait()를 call한다. 이 때 함수의 parameter은 child thread의 id이다. 이 때 semaphore을 이용한다. semaphore child\_lock은 sema\_down()되어, child thread가 process.c의 process\_exit()에서exit될 때 sema\_up해주기 전까지 잠긴다. 이렇게 함으로써 wait의 mechanism을 구현할 수 있다.

-READ: parameter로 file descriptor, 읽고자 하는 buffer, 읽고자 하는 크기 size가 온다. file descriptor가 0일 때 READ를 실행한다. size만큼, 혹은 중간에 ‘\0’이 나올 때까지 읽어준 후, 읽은 글자 수를 return한다.(즉 f->eax에 할당한다.)

-WRITE: parameter로 file descriptor, 쓰고자 하는 buffer, 쓰고자 하는 크기 size가 온다. file descriptor가 1일 때 WRITE를 실행한다. putbuf(buffer, size)를 call하여 size만큼 buffer을 써준다. 이후 쓴 글자 수를 return한다.(즉 f->eax에 할당한다.)

1. Additional System calls

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

-pintos/src/lib/user/syscall.c에 parameter가 4개일 때, 즉 syscall4를 구현

-…/lib/syscall-nr.h에 새로운 system call을 맡는 system call number 지정

-pintos/src/userprog/syscall.c의 system handler에서 새로운 system call의 함수 구현

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

**10,15,11,13 입력결과 55,15로 로 잘 뜨는 것을 확인할 수 있다.**

