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

담당 교수: 김영재

조 / 조원: 20171608 김 민

개발 기간: 9/11 ~ 9/30

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

pintos에서 argument passing, syscall call handler, additional system call을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

User application에 입력된 argument가 x86 convention에 맞게 stack에 저장된다.

1. User Memory Access

instruction에서 접근하려고 하는 memory가 user 영역인지, kernel 영역인지를 구분하여 kernel 영역의 메모리를 protect 할 수 있게 된다.

1. System Calls

Argument passing으로 전달된 parameter를 이용해 적절한 system call을 호출하여 올바른 실행결과를 보일 수 있도록 한다.

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

Load 함수에서 void \*\* esp 변수를 이용하여 x86 convention에 맞게 parameter를 역순으로 stack에 적재, parameter가 적재된 address 또한 역순으로 stack에 적재, parameter의 address의 시작점을 적재, parameter의 개수인 argc를 적재, return address를 적재한다.

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

pintos에서 memory는 user application 영역과 kernel 영역으로 나뉘어 있는데 user application에서 실행한 instruction이 kernel 영역의 memory에 접근하게 되면 kernel에 심각한 손상이 발생할 수 있게 된다.

* + Invalid memory access를 어떻게 막을 것인지 설명

vaddr.h 파일에 해당 address가 user application의 영역인지, kernel 영역인지 확인 할 수 있는 is\_user\_vaddr() 함수와 is\_kernel\_vaddr() 함수가 존재한다. 이 함수를 이용하여 memory에 접근하기 전에 올바른 영역의 memory인지 확인한다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

User application은 kernel의 memory 영역에 접근할 수 없도록 protect 되어 있기 때문에 kernel에서 제공하는 기능을 직접적으로 수행할 수 없다. 따라서 user application이 kernel에서 제공하는 기능을 이용할 수 있도록 도와주는 interface가 필요한데, 이것이 system call이다.

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

Halt: pintos system을 종료한다.

Exit: process를 종료하면서 process가 점유하고 있던 자원을 반환한다.

Exec: 전달받은 parameter에 해당하는 프로그램을 수행하는 새로운 process를 생성한다.

Wait: 부모 process가 자식 process가 종료될 때까지 기다릴 수 있도록 한다.

Read: STDIN에서 입력을 받아 buffer에 저장할 수 있도록 한다.

Write: STDOUT에 buffer에 담긴 내용을 출력한다.

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

User level에서 system call API를 하면 interrupt가 발생해 제어권이 kernel에게 넘어가게 된다. Kernel은 stack에 담긴 내용을 토대로 적절한 system call을 실행시키고 system call의 실행이 종료되면 user application으로 다시 제어권이 넘어가게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

9/11 ~ 9/15: 제공된 code 파악 및 명세서, manual 숙지, argument passing 구현

9/16 ~ 9/22: argument passing 구현, system call handler 구현, user memory access 구현, 일부 system call 작성(halt, exit, read, write)

9/23 ~ 9/30: 남은 system call 작성 및 구현된 system call 수정, 전체적인 코드 수정

* 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.c의 load() 함수 내부에 Argument passing을 위한 문자열 처리, stack을 쌓기 위한 함수 구현

Vaddr.h를 참고하여 User memory access 관리를 위한 함수 구현

Sycall.c의 syscall\_handler()에서 switch case문을 이용해 system call을 handle, 각 system call에 알맞은 기능을 수행하기 위한 함수 작성

Wait의 구현을 위해 thread.h의 structure thread에 semaphore 추가 및 child thread를 저장하기 위한 list 추가, thread.c의 init\_thread() 함수에 해당 내용 반영, process\_exit, process\_wait에서 적절한 semaphore의 조작

오류가 발생하는 test case를 위한 코드 추가(exception.c의 page\_fault(), process.c의 process\_execute() 등)

1. **연구 결과**
   1. **Flow Chart**

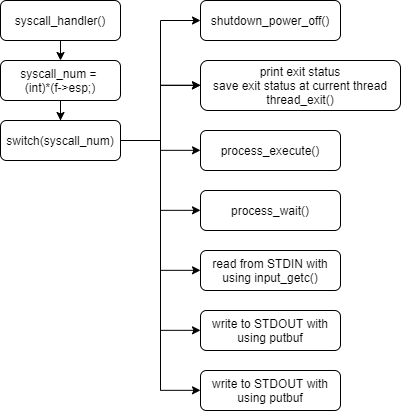
* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

**텍스트이(가) 표시된 사진

자동 생성된 설명**

1. User Memory Access



1. System Calls

텍스트이(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

Load 함수에서 push\_arg()라는 함수를 만들어 stack에 argument를 x86 convention에 맞게 적재하였다. 전달받은 file\_name에서 argc의 값을 구하고 각 parameter를 tokenize하여 stack에 적재하면서 해당 esp의 값을 저장하여 parameter가 stack에 저장된 위치 또한 stack에 적재한다. 다음으로 argument의 시작 주소를 저장하고(esp – 4) argc의 값과 return address를 stack에 적재한다. 각 값을 적재할 때 pintos는 multi-thread system이라는 점을 감안하여 thread safe한 strlcpy 함수를 사용하였다.

1. User Memory Access

vaddr.h 함수에서 제공하는 is\_user\_vaddr() 함수를 이용하여 system call에서 사용하는 parameter의 값을 검증하는 valid\_check() 함수를 구현하였다. valid\_check() 함수는 넘겨받은 값이 user 영역의 주소가 아니라면 exit(-1)을 호출하도록 되어 있다. 추가로 page fault가 발생한 경우 적절한 예외처리를 위해 exception.c의 page\_fault() 함수에서 flag와 is\_kernel\_vaddr() 함수를 사용하였다.

1. System Calls

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

Syscall.c에서 syscall\_handler() 함수 내부에 switch case문을 이용하여 src/lib/sycall-nr.h에 enum으로 정의된 system call의 번호에 맞게 각 system call을 수행할 수 있도록 구현하였다.

Halt의 경우 명세서를 따라 shutdown\_power\_off()를 호출하였다.

Exit의 경우 pintos의 채점 요구에 따라 exit status를 출력하며 process\_exit()를 호출하였다. 또한 이후 구현되는 wait을 위해 exit status를 current thread에 저장한다.

Exec의 경우 명세서를 따라 process\_execute()를 호출하였다.

Wait의 경우 process\_wait()을 호출하는데, process\_wait()에서는 자신의 child thread중 wait의 대상이 되는 thread가 있는지 확인하고 존재한다면 sema\_down()을 이용하여 child thread의 종료를 기다린다. Child thread는 process\_exit()에서 자신을 종료하며 sema\_up()을 이용해 parent thread를 깨워준다. 여기서 parent thread가 child\_list에서 해당 child를 remove하기 위해 list\_remove() 함수를 호출하는데 이때까지 child는 memory상에 남아 있어야 하기 때문에 sema\_down()을 이용하여 parent thread가 list\_remove()를 마치고 다시 sema\_up() 통해 자신을 깨워줄 때까지 기다린다. 이러한 과정을 위해 thread.h에 정의된 struct thread에 child\_list, child\_wait, mem\_wait 등의 멤버를 추가해주었다.

Read의 경우 1번 project에서는 STDIN으로부터의 입력만을 처리하면 되므로 STDIN에 해당하는 file descriptor에 대해 input\_getc()를 사용하여 read를 구현하였다.

Write의 경우 또한 STDOUT으로의 출력만을 처리하면 되므로 STDOUT에 해당하는 file descriptor에 대해 putbuf()를 사용하여 write를 구현하였다.

1. Additional System calls

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

새로운 system call을 test 하기 위한 example을 작성한 뒤 Makefile을 수행하여 예제가 compile 될 수 있게 하였다. 또한 lib/user/syscall.c에 4개의 argument를 받는 system call을 처리할 수 있도록 syscall4() 함수를 작성하였고, 새로운 system call을 호출하기 위한 함수(Fibonacci(), max\_of\_four\_int()) 작성하였다. 추가로 syscall-nr.h에 새로운 system call의 번호를 정의하기 위해 enum을 추가로 작성하였다. Syscall.c에서는 넘겨받은 parameter를 이용하여 기대되는 값을 return하여 eax register에 저장하였다.

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

텍스트이(가) 표시된 사진

자동 생성된 설명