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

담당 교수 : 박성용

조 / 조원 : 20181693 최원빈

개발 기간 : 2022.09.18 ~2022.10.03

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**
* pintos에서 여러 system call 명령어들을 수행할 때 들어온 명령어를 적절하게 나누고, 스택에 쌓고, 호출하여 핸들러까지 가게 한 후, 궁극적으로 작업이 우리가 의도한대로 완료된 후 정상적으로 종료하게 하는 것이 이번 프로젝트의 목표이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

들어온 argument들을 적절하게 parsing해(띄어쓰기 단위), pintos manual의 기준에 맞춰 esp 스택에 쌓아 보낸다.

1. User Memory Access

접근하려는 address가 valid한 address인지 확인하고 접근할 수 있게 된다.

1. System Calls

현재 작동하지 않는 여러 system call들 중 일부(halt, exit, exec, wait, read(stdin), write(stdout))를 정상적으로 실행하고 종료될 수 있게 된다.

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

먼저 file\_name으로 들어온 argument를 띄어쓰기 단위로 parsing한다. 이후 esp 스택에 쌓는데 그 순서는 나중에 들어온 argument 부터 차례대로 넣고, 이후 필요하다면 word-align을 해준다(0을 넣음). 이후 먼저 넣은 argument들의 주소를 넣어주고, 전체의 주소를 넣는다. 그런 다음 argument의 개수와 return address를 넣는다.

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

User mode에서는 kernel 영역에 해당하는 주소에 접근하면 안 된다. 권한없이 kernel 영역의 address에 접근하게 되면 invalid memory access가 발생하게 된다.

* + Invalid memory access를 어떻게 막을 것인지 설명

Pintos/src/threads/vaddr.h를 include 하여 해당 헤더에 있는 is\_user\_vaddr을 이용하여, 의심스러운 주소를 사용하기 전에 먼저 valid한지 확인할 것이다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

시스템 콜은 운영체제가 제공하는 서비스에 대한 접근방법을 제공한다. 프로그램을 작성하다 보면 user level 함수들만으로는 수행하기 힘든 작업들이 있기에 kernel의 도움은 필수적이고, 이 커널 모드에서 작업을 수행하기 위해 시스템 콜을 호출한다. 다시 말해 시스템 콜은 user mode program이 커널 기능을 사용할 수 있게 하고, 이때 시스템 콜은 kernel mode에서 실행되고 처리 후 user mode로 복귀한다.

* + 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 종료시킨다.
    - exit : 현재 프로세스를 종료시킨다. 종료한 stauts도 함께 저장한다.(정상적으로 종료시 0)
    - exec : 자식 프로세스를 생성하고 프로그램을 실행시킨다. 성공 시 생성된 프로세스의 pid 값을 반환한다. 이때 부모 프로세스는 자식 프로세스의 프로그램이 메모리에 적재될 때까지 대기한다.
    - wait : 자식 프로세스가 종료될 때까지 대기한다.
    - read(stdin) : 열린 파일의 데이터를 읽는다. stdin의 file descriptor 가 0이므로 fd==0 인 경우 다 읽은 후 읽은 byte 수를 반환 한다.
    - write(stdout) : 열린 파일의 데이터를 기록한다. stdout의 file descriptor 가 1이므로 fd==1 인 경우 buffer 를 putbuf()를 통해 출력한 후 size를 return 한다.
    - fibonacci : 피보나치 수열의 n 번째 수를 return 한다.
    - max\_of\_four\_int : 4가지 정수 중 가장 큰 수를 return 한다.
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

시스템 콜 명령어가 입력으로 들어오게 되면, pintos/src/lib/user/syscall.c에 있는 해당 함수들이 매크로로 구현되어 있는, syscall을 수행한다. 이때 int $0x30으로 kernel 영역의 interrupt vector table 중 syscall\_handler를 호출한다. 이후 우리가 작성한 syscall\_handler에서 함께 보낸 이미 정의해둔 syscall number에 따라 esp에 쌓인 넘어온 argumnet들로 작업을 수행한후 return하여 user level로 돌아온다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

09.18 ~ 09.25 사전 개념 복습 및 기존 제공된 코드 체크, pintos manual 참고.

09.26 ~ 09.28 argument parsing 및 argument passing 구현.

09.28 ~ 09.30 system call handler 작성 및 user memory access 체크 완료.

10.01 ~ 10.03 추가 기능 구현 및 코드 보완.

* 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

pintos/src/userprog/process.c의 load 함수에서 file\_name을 parsing 하는 부분을 추가할 것이다. 이후 esp 스택에 쌓는 부분을 추가할 것이다.

* User Memory Access

pintos/src/userprog/syscall.c에서 check\_user\_addr() 함수를 만들어 어떤 주소를 사용하기 전에 먼저 valid 한 address인지 확인할 것이다

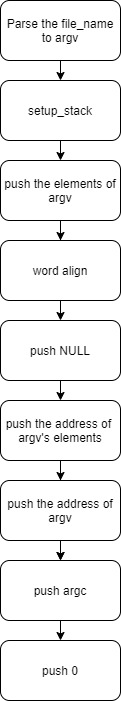
* System Calls

pintos/src/userprog/syscall.c에 들어올 수 있는 여러가지 intr\_frame 에 대하여 switch 문을 통하여 case 를 나누고 각각의 함수를 호출한다. 위에서 언급했던 halt, exit, exec, wait, read(stdin), write(stdout) 과 fibonacci, max\_of\_four\_int 를 추가할 것이다.

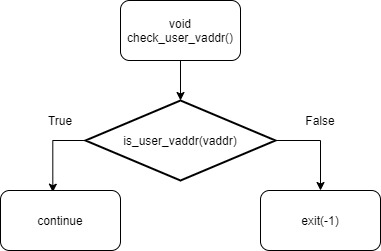
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

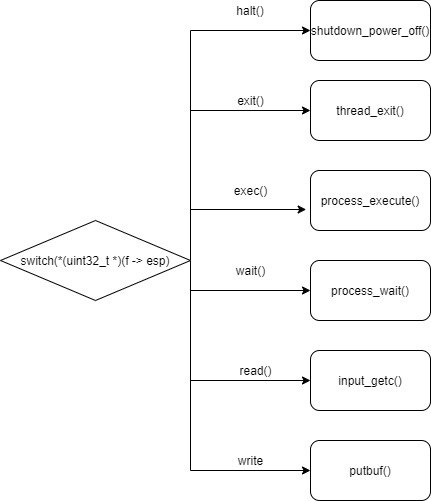
1. Argument Passing



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

Pintos/src/usr/process.c의 load 함수에서 argument parsing을 해준다. 이때 argc에는 온 argument 개수를, argv에는 각 띄어쓰기 단위로 나눈 argument들을 ‘\0’를 붙여 차례대로 저장한다. 이때 multi-thread 프로그램에서도 안전한 strtok\_r 함수를 사용하였고, 이 함수는 원본 문자열의 값이 바뀔 수 있기에 안전하게 temp에 복사하여 사용하였다. 이후 esp 스택에 넘겨줘야 할 정보들을 쌓는데 그 순서는 위에서 언급한대로 \*argv 원소들, (필요시)word align, NULL, \*argv 원소들의 주소, argv의 주소, argc, return address이다. 이때 맨처음에 명세서를 보지 않고 word-alignment를 위한 자리와, NULL을 위한 자리, 그리고 return address에 잘못된 값을 넣어 정상적으로 처리가 되지 않아 수정했었다.

1. User Memory Access

System call을 수행하기 위해 핸들러를 작성하는 코드(pintos/src/syscall.c)에서 사용할 address가 valid한지 확인하기 위해, 사용하기 전에 미리 check\_user\_addr()을 써서 확인한다. check\_user\_addr는 threads/vaddr.h에 있는 is\_user\_vaddr 을 이용해 만약 아닐시 exit(-1)하여 invalid memory access를 방지한다.

1. System Calls

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

기본적으로 모든 시스템 콜들은 syscall-nr.h에 있는 순서에 따라 고유번호가 존재하고 그에 따라 syscall\_handler에서 switch 문을 통해 case 별로 처리한다.

* Halt : proto type은 void halt (void)이다. shutdown\_power\_off()를 호출해 pintos를 강제 종료한다.
* Exit : proto type은 void exit (int status)이다. 현재 thread\_current()의 status를 넘어온 인자 status로 하고, 그 쓰레드의 이름과 status를 출력하고, thread\_exit()을 한다. 이때 맨 처음 출력할 때는 echo 기준 echo x : exit(0)이 출력되는데, 이는 file\_name 전체가 넘어가서 그랬다. 때문에 여기(thread\_excute)에서도 parsing을 수행하고 첫 인자만 따로 저장한 argv을 넘겨주면 알맞게 출력된다.
* Exec : proto type은 pid\_t exec (const char\* cmd\_line)이다. 인자로 넘어온 cmd\_line 을 수행하도록 process\_execute(cmd\_line)를 호출하고 그 리턴값을 리턴한다.
* Wait : proto type은 int wait (pid\_t pid)이다. 자식프로세스가 종료될 때까지 대기한다. 이때 synchronization 기법을 이용한다. 자식의 exit status를 알려면 부모는 자식의 쓰레드를 알고 있어야 하므로, thread 구조체에 관련 원소들을 추가해준다. construct semaphore child\_lock, struct list child\_list, struct list\_elem child\_elem, int exit\_status를 추가하고 thread.c 의 init\_thread 부분에 위에 추가한 변수들을 initialize 하는 코드도 작성한다. 이후 process\_wait()을 수정한다. thread\_current()의 child\_list를 돌아보면서 내가 기다리고 있는 child\_tid 가 맞다면 sema\_down()을 통해 child 가 끝날 때까지 기다린다. 이후 child가 process\_exit()할 때 sema\_up()을 해주게 한다면 child\_tid가 끝날 때까지 wait 할 수 있게 된다. 추가적으로 하나의 lock을 더 잡는데, process\_exit을 할 때 child\_tid 가 죽는 경우 list\_remove를 못 하는 경우를 막기 위해서이다. Mem\_lock을 하나 더 정의해 wait과 exit에서 잡고 풀면 부모가 자식쓰레드를 잘 기다리게 된다.
* Read : proto type은 int read(int fd, void \*buffer, unsigned size)이다. 이번 프로젝트에서는 stdin에서 들어온 경우만 생각하면 되므로 fd 는 0일 때, input\_getc()를 리턴한다. 이때 리턴값이 eax에 저장되어야 돌아갈 수 있으므로, halder를 작성할 때 함수의 리턴값이 f->eax에 저장되게 한다.
* Write : proto type은 int write(int fd,const void \*buffer, unsigned size)이다. 이번 프로젝트에서는 stdout으로 출력하는 경우만 생각하면 되므로 fd는 1일 때, putbuf(buffer, size)를 해주고 size를 리턴한다. 이때 read와 마찬가지로 리턴값이 eax에 저장되어야 돌아갈 수 있으므로, halder를 작성할 때 함수의 리턴값이 f->eax에 저장되게 한다.

1. Additional System calls

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

Pintos/src/lib/syscall-nr.h에 열거된 syscall number에 SYS\_FIB(13)와 SYS\_MOFI(14)를 추가하였다.

Pintos/src/lib/user/syscall.h에 Fibonacci, max\_of\_four\_int의 prototype을 추가하였다.

Pintos/src/lib/user/syscall.c에 Fibonacci, max\_of\_four\_int가 불렸을 때 syscall을 호출하도록 하게하였다. 이대 max\_of\_four\_int에서 사용할 syscall4를 새로 매크로로 정의하였다.

Pintos/src/examples에 additional.c를 생성한다. Additional.c는 실행과 함께 들어온 argument를 정수로 바꿔서 fibonacci와 max\_of\_four\_int의 parameter로 넘겨 호출하고 출력한다. 이때 string에서 정수로 바꿔주는 convert 함수를 직접 작성하여 사용하였다. 이후 Makefile에 additional.c도 추가하여 make 입력시 additional이라는 excutable file이 생성되게 한다.

Pintos/src/userprog/syscall.h에 prototype을 추가하였다.

Pintos/src/userprog/syscall.c의 switch문에 SYS\_FIB와 SYS\_MOFI일 경우를 추가하고 각각 esp에 넘어온 인자들을 valid한지 검사하고, 자체적으로 작성한 Fibonacci, max\_of\_four\_int 함수에 넘기고 f->eax 레지스터에 넣어 반환되도록 하게 한다.

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

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**