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

담당 교수 : 김영재

조 / 조원 : 김준우

개발 기간 : 9/27 ~ 10/7

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

운영체제의 첫번째 project로 개발해야 하는 목표는 pintos manual상의 project2의 내용이다. 입력 받은 argument를 parsing하고, user stack에 올바르게 저장한 후 system call handler를 호출해 system call handler가 알맞은 system call을 수행하도록 구현해야 한다. 그리고 이 과정에서 유효하지 않은 포인터의 접근을 막아주는 코드를 작성해야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

* Argument Passing을 구현하면, 입력 받은 argument를 공백을 기준으로 parsing 되어 user stack에 저장된다. hex\_dump() 함수를 이용하면 pintos에서 정의 된 오른쪽 인자부터 왼쪽으로 스택에 쌓이는 규칙에 맞게 user stack에 저장된 것을 확인할 수 있다.

1. User Memory Access

* User Memory Access를 구현한 결과는, pintos 상에서 user address와 kernel address영역은 분리되어 있고 user가 kernel에 접근할 수 없다. 따라서 user가 kernel에 접근 하는지 확인하고 막아준다.

1. System Calls

* System call을 구현한 결과는 user가 호출을 하면 kernel에 접근하여 system call에 알맞은 코드를 수행하고 user에 return 해준다.
  1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing
  + 커널 내 스택에 argument를 쌓는 과정 설명

Stack에 argument를 쌓기 위해선 입력 받은 argument를 공백을 기준으로 Parsing을 해줘야 한다. 공백으로 parsing된 argument들은 stack에 순서대로 ‘\0’까지 모두 push 해준다. ‘\0’까지 stack에 넣어주는 이유는 다른 argument와 구분을 해주기 위함이다. 이렇게 stack에 모두 넣고 나서 word 사이즈를 맞춰 주어야 하기 때문에 padding을 넣어서 크기를 맞춰준다. 그 후에 주소값를 넣어줘야 한다. 따라서, 데이터 영역과의 구분이 필요하기 때문에, padding을 모두 추가하고 나서 NULL 포인터를 하나 추가로 넣어줘 구분한다. 그렇게 모든 주소를 stack에 추가하고 나서 argv의 주소까지 추가 후 argc(parsing된 argument의 개수)와 return address를 stack에 추가하면 passing이 완성된다.

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

Pintos 상에서는 kernel 영역과 user 영역이 구분되어 있고 user가 user의 address 영역에만 접근할 수 있고, kernel address 영역에 접근할 수 없다. 따라서 invalid memory access란 개념은 user가 kernel 영역에 접근하는 것이다.

* + Invalid memory access를 어떻게 막을 것인지 설명

Invalid memory access를 막기 위해 pintos에 선언 되어 있는 함수인 is\_user\_vaddr(), is\_kernel\_vaddr() 함수를 이용해 user address 영역이 kernel address 영역을 point하고 있는지를 확인 할 수 있다. 이 두 함수를 이용해 만약 user가 kernel address 영역을 point 하고 있다면, 해당 thread를 종료 시켜 invalid memory access를 막을 것이다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

System call은 앞에서 설명한 invalid memory access 때문에 필요하다. Kernel address 영역에 존재하는 함수를 user가 실행해야 하는 경우, invalid memory access이므로 block 된다. 이때 system call 이 user와 kernel을 연결해주는 역할을 해서 kernel에 있는 함수를 user가 이용할 수 있게 하기 때문에 필요하다.

* + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
* halt() : pintos상에 있는 shutdown\_power\_off() 함수를 호출하여 pintos 실행을 종료하는 함수다.
* exit() : 현재 실행되고 있는 user의 process를 process\_exit()함수를 호출해 종료하는 함수다.
* exec() : 새로운 child process를 process\_excute()함수를 호출해 만들어서 execute 해주는 함수다.
* wait() : parent process가 child process가 종료될 때까지 기다리게 하기 위한 함수로 child process의 tid가 유효하다면 child process가 종료될 때까지 기다리고 child\_process가 종료되면 exit\_status를 return해주는 함수다.
* read() : file descripter로부터 buffer의 내용을 읽어오는 함수다. STDIN으로 들어온 내용만을 읽어온다.
* write() : file descripter에 buffer의 내용을 써주는 함수다. Read와 마찬가지로 STDOUT에만 내용을 써준다.
* Fibonacci() : argument로 입력 받은 인자 n에 대해서 n번째 Fibonacci 수열의 값을 구해서 return 해주는 함수이다.
* Max\_of\_four\_int() : argument로 입력 받은 인자 a,b,c,d에 대해서 4개중 가장 큰 int를 구해서 return 하는 함수다. 4개의 인자를 입력 받아야 하기 때문에 syscall4를 lib/user/syscall.c에 구현하여 개발한다.
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

User가 systemcall을 호출하면 userprog/syscall.c에 있는 syscall\_handler()가 호출된다. Syscall\_handler()는 인자로 스택의 주소를 가진 struct intr\_frame 를 받는다. 여기엔 스택 포인터인 esp와 systemcall의 return값을 받을 register인 eax가 선언되어 있다. Esp를 통해 어떤 system call을 호출할지 알아내고, 알맞은 함수가 실행되어 kernel의 함수를 user가 이용하게 되고 return 값을 eax에 저장하도록 해 다시 kernel에서 user로 돌아오게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

9/27~9/30 : 공백을 기준으로 argument parsing과 스택에 쌓는 argument passing을 구현한다. Hex\_dump를 통해 확인해준다.

10/3 : system call handler와 halt, exit, exec,wait, read, write함수, additional system call의 함수 구현

10/4 ~ 10/5 : system call 번호와 syscall4 구현. Thread 구조체에 child thread의 정보를 저장할 정보 추가. Process\_execute, process\_wait, process\_exit 에 thread가 safe하게 돌아가도록 추가

10/6~10/7 : make check를 돌려보며 잘못 작성된 부분 고치고 확인, 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)**
* Argument passing은 load 함수에 공백을 기준으로 나누는 parsing 부분을 작성 passing은 새로운 함수를 선언해서 작성, 이 함수에 hex\_dump를 호출에 스택에 쌓이는 것을 확인
* Userprog/syscall.c syscall\_handler 와 각 system call에 알맞은 함수들 추가, lib/syscall-nr.h 에 추가한 system call에 대한 번호 추가, lib/user/syscall.c 와 syscall.h에 fibonacci와 max\_of\_four\_int에 대한 함수 추가, 4개의 인자를 받을 수 있는 syscall4를 추가. Examples/additional.c 에 fibonacci와 max\_of\_four\_int를 사용할 수 있도록 추가. Examples/Makefile에 additional.c의 실행파일을 실행할 수 있도록 추가
* Threads/thread.h에 thread구조체에 child thread와 관련된 요소 추가. Threads/thread.c init\_thread에 추가한 요소도 init 하도록 추가. Userprog/process.c에 process\_execute, process\_wait, process\_exit에 thread에 관련된 함수 추가.

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

스크린샷, 텍스트, 도표, 영수증이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

스크린샷, 도표, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명

1. System Calls

텍스트, 영수증, 도표, 평행이(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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명userprog/process.c

Argument를 parsing하는 부분이다 load함수에 구현해주었다. 우선 원래의 file\_name이 변경되면 안 되기 때문에 이를 복사한 fn\_copy를 선언해주었다. 여기서 malloc을 이용해 메모리 공간을 할당 해줬는데 이는 thread safe하게 만들어 주기 위함이다. 그 후 strlcpy함수를 통해 fn\_copy에 file\_name을 복사해주고 strtok\_r 함수를 통해 공백을 기준으로 parsing 해주었다. 이때 ptr를 while을 통해 끝까지 탐색해야 하기 때문에 char 포인터인 parsed\_fn을 선언해 전체를 탐색하기 전에 가장 첫번째 위치를 저장해준다. 그리고 while문을 돌면서 argv에 parsing된 argument를 저장해주고 argument의 개수인 argc를 카운트 해준다. 이렇게 parsing된 argument를 file을 열 때 인자로 이용해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 웹사이트이(가) 표시된 사진

자동 생성된 설명

Parsing된 argument를 stack에 쌓는 함수다. 스택에는 parsing된 가장 마지막 인자부터 쌓아줘야 하기 때문에 argc-1부터 반복문을 이용해 stack에 쌓아준다 word의 크기가 4이기 때문에 esp – 4를 해주면서 값을 넣어준다. 그리고 데이터가 스택의 어디에 저장되어 있는지 주소값도 데이터를 모두 stack에 쌓아주고 나서 쌓아줘야 하기 때문에 addr에 저장을 해준다. 첫번째 반복문이 끝나고 나서 word의 크기를 맞춰 줘야 하기 때문에 padding을 이용해 4로 나눈 나머지가 0이 아닌 경우 그 차이만큼 esp의 값을 빼주어서 word의 크기를 맞춰 줬다. 그리고 데이터 영역과 주소 영역의 구분을 위해 NULL을 추가해준다. 그리고 주소값을 stack에 추가하고 그 후로 순서대로 argv의 주소값, argc 값, 다른 argument와의 구분을 위한 NULL까지 stack에 쌓아주고 함수가 종료된다.

1. User Memory Access

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Userprog/syscall.c

스택포인터인 esp를 이용해 어떤 system call을 원하는지 확인한 다음 인자의 개수 만큼 invalid memory access를 확인하기 위한 새로운 함수인 check\_user\_vaddr을 통해 user가 kernel 영역을 point하고 있는지 확인해 그렇다면 exit함수를 통해 종료해주는 코드다.

1. System Calls

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

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**userprog/syscall.c

* Halt() : shutdown\_power\_off() 를 호출해 pintos를 종료한다.
* Exit() : 종료한다는 메세지를 출력하고 thread의 exit\_status를 인자로 받은 status로 업데이트 해준다. 그리고 thread\_exit() 함수를 호출해 thread(child process)를 종료해준다.
* Exec() : thread를 만들고 실행해주는 process\_execute 함수를 호출해 실행해준다.
* Wait() : process\_wait(tid) 를 호출해 thread가 종료될 때까지 기다리게 한다.
* Read() : file\_discripter으로부터 buffer를 읽어와야 한다. 이때 fd가 STDIN인 경우에만 읽어와야 하기 때문에, fd == 0 이면 반복문을 통해 buffer를 읽어온다. 읽어 오는 것은 input\_getc 함수를 이용해 읽어오고 ‘\0’이 입력되면 반복문을 종료하고 buffer의 크기를 return한다. Fd == 0이 아니면 -1을 return 해서 오류임을 알려준다.
* Write() : file\_discripter에 buffer를 써줘야 하는데 read와 마찬가지로 STDOUT일때만 써줘야 하기 때문에 fd==1 인 경우에 putbuf를 통해 buffer를 써주고 buffer의 크기인 size를 return 해준다. Fd == 1이 아닌 경우엔 buffer를 써주면 안 되기 때문에 -1 을 return 해 오류임을 알려준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Userprog/process.c process\_execute

System call의 exec 함수가 호출하는 부분이다. Thread를 생성하고 실행하기 위해 만들어진 함수로 Thread에서 실행하기 위해서 file\_name을 parsing하여 thread를 create 해준다. Thread safe하게 실행하기 위해서 malloc을 통해 동적할당을 해서 file\_name을 copy해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

System call의 wait 함수에서 호출되는 함수다. Child thread가 실행 되고 있으면 parent process가 기다려 주도록 반복문을 돌리고, child thread의 실행이 종료되면 thread의 exit\_status를 갱신하고 exit\_status를 return해준다.

Userprog/process.c process\_wait

텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

Userprog/process process\_exit

이렇게 thread와 관련된 코드를 추가해 thread에서 system call이 작동하도록 해준다.

1. Additional System calls

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

**텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명**userprog/syscall.c

우선 추가한 system call인 max\_of\_four\_int와 Fibonacci의 함수 구현 파트다. Max\_of\_four\_int의 경우 인자로 입력 받은 4개의 정수 중 최대값을 return 해줘야 하기 때문에 max라는 새로운 int형 변수를 선언해 a를 넣어주고 if문을 통해 남은 3개의 인자 b,c,d 중에 더 큰 것이 있는지 확인해 있다면 max값을 갱신해주고 max를 return하여 구현해주었다.

Fibonacci의 경우 Fibonacci 수열의 n번째 값을 return 해줘야 한다. Fibonacci 수열을 순서대로 써보면 0,1,1,2,3,5,8… 을 알 수 있다. 이 때문에 n이 0,1,2를 입력 받은 경우는 예외처리를 통해 return을 해주고 그 이상의 숫자는 반복문을 통해 n번째 Fibonacci 수열의 값을 구하여 return 해준다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명lib/syscall-nr.h

새로운 system call이 생겼기 때문에 handler를 통해 컨트롤 해주기 위해 system call number를 정의해주었다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명lib/user/syscall.c

Syscall4 는 max\_of\_fout\_int가 4개의 인자를 받는 함수인데 syscall.c에 4개의 인자를 받는 함수에 대한 것이 정의되어 있지 않기 때문에 새로 정의해서 4개의 인자를 받는 system call을 작동 할 수 있게 해주었다.

Max\_of\_four\_int는 syscall4를 이용하고, fibonacci는 하나의 인자만 가지기 때문에 syscall1을 통해 구현해주었다.

* 1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**
* pintos --filesys-size=2 -p ../examples/additional -a additional -- -f -q run ‘additional 10 20 62 40’
* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명