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

담당 교수 : 김영재

조 / 조원 : 20171669 이재영

개발 기간 : 2021.09.13 – 2021.10.01

1. **개발 목표.**

Pintos 환경에서 user program을 실행시키는 것이 Project1의 목표이다. 입력 받은 argument를 적절히 잘라내 user stack에 쌓는 것과 더불어 user program에서 system call이 제대로 실행되도록 하는 것을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Argument Passing

argument의 실제 실행되어야 할 명령어 부분이 load되고 parsing한 argument들을 stack에 쌓아서 프로그램을 수행할 때 stack을 통해 접근할 수 있게 해준다.

1. User Memory Access

User memory에 access할 때, 전달받은 주소 값이 적절한 공간을 사용하는지 확인하는 작업을 수행하며 적절하지 못한 공간(user space가 아니거나 kernel space를 사용할 때는 system call에서 구현한 sys\_exit 함수로 exit(-1) 작업을 수행한다.

1. System Calls

/pintos/src/userprog/syscall.c의 syscall\_handler() 함수에서 halt, exit, exec, exit, write, read 명령어를 구현해서 각각의 맞게 프로그램이 수행되도록 한다. 추가적인 system call로 fibonacci와 max\_of\_four\_int를 구현한다.

* 1. **개발 내용**
* Argument Passing

Setup\_stack(esp) 로 stack을 setup한 후 argument를 stack에 쌓는 작업을 수행한다. esp에 \0(null값)을 포함한 argument의 길이만큼 감소시킨 후 스택에 순차적으로 쌓는다. 이 때, 변수 cnt는 parse된 argument의 개수로 cnt만큼 argument의 길이를stack pointer(\*esp)에서 감소시킨 후 \0(null) 값을 포함한 argument값을 넣는다. 현재 stack pointer의 값(주소 값)을 address 배열에 저장해 놓는다. 이 때, word alignment를 생각해서 변수 align\_num에 더한다. Word alignment를 구현하는 방법은 4-B에서 코드를 통해 설명한다. Argument의 값들을 전부 stack에 쌓았으면 word alignment을 맞추기 위해 4-align\_num을 감소시키고 나머지 4byte를 \*esp에서 감소시킨 다음 0값을 넣는다. Argument의 개수만큼 저장해 놓은 각각 argument가 쌓인 stack의 주소 값을 쌓는다. 이 후, Argv의 주소 값, argument의 개수의 주소 값, return address를 쌓는다. 쌓이는 코드에 대한 자세한 설명은 4-B에서 한다.

* User Memory Access

Physical memory는 virtual address에 mapping되어 있다. 이 때, physical address 0은 PHYS\_BASE에 access 가능한데 address가 PHYS\_BASE보다 크거나 같은 경우는 kernel 영역에 접근하는 것으로 invalid한 memory access로 볼 수 있다.

/pintos/src/threads/vaddr.h에 정의되어 있는 is\_user\_vaddr, is\_kernel\_vaddr 함수를 사용하여 user 영역이 아닌 kernel영역을 사용하는 경우의 예외처리를 진행한다.

* System Calls

System call을 통해 user와 kernel영역을 매개할 수 있기 때문에 system call이 필요하다. User level에서 load함수를 통해 parse한 시스템 콜 명령어를 file\_sys를 통해 open을 하면서 pintos/src/lib/user의 system call.c에 정의되어 있는 system call 함수에 선언되어 있는 asm code, syscall1,2,3를 호출하면서 stack에 각각의 system call number와 arguments, interrupt handler를 stack에 push해준다. 그 후에 pintos/src/userprog의 syscall.c의 syscall\_handler를 호출해서 intr\_frame f를 통해 user stack에 접근해서 각각의 system call에 맞는 동작을 수행해 kernel의 API를 수행한 후 user로 돌아온다.

halt : Pintos를 종료하는 system call이다.

exit : 현재 thread를 exit하는 system call이다.

exec: child process를 create해서 execute하는 system call로 process.c에 구현되어 있는 process\_execute함수를 사용하여 create한 child process의 tid를 return 받는다.

wait: child process끝날 때까지 기다리는 system call로 process.c에 구현되어 있는 process\_wait함수를 사용한다. process\_wait에서는 thread를 탐색해서 child thread가 죽을 때까지 탐색한 후 죽었을 때의 thread의 exit status를 반환한다.

write: STDIN으로 쓰는 동작을 구현하는 system call이다.

read: STDOUT으로부터 읽어들이는 동작을 구현하는 system call이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

9/13 ~ 9/16 : Argument를 parsing하고 stack에 쌓는 작업

9/17 ~ 9/24 : System call(halt, exit, wait, exec, read, write) 구현

9/25 ~ 9/28 : user program에 제대로 접근하지 않는 것을 제외시키기(User Memory Access) + Additional System call(fibonacci, max\_of\_four\_int) 구현

9/29 ~ 10/1 : Project1 보고서 작성

* 1. **개발 방법**
  + Argument Passing

</pintos/src/userprog/process.c>

Load 함수에서 공백을 기준으로 parse해주기 위해 /pintos/src/lib/string.c에 정의되어 있는 strok\_r 함수와 string의 길이를 반환해주는 strlen함수를 사용했다.

* + User Memory Access

</pintos/src/userprog/syscall.c>

Argument의 주소가 user memory인지를 확인하기 위해 is\_user\_vaddr함수를 사용했다. User memory 영역에 있지 않다면 sys\_exit(-1)을 호출해서 thread를 exit시켰다.

</pintos/src/userprog/exception.c>

Page\_fault함수에서 user가 0, 즉 false일 때, 혹은 address가 kernel영역에 있는 경우(is\_kernel\_vaddr로 확인할 수 있다) userprog/syscall.c에 정의되어 있는 sys\_exit(-1)을 호출해서 thread를 exit시켰다.

* + System call

</pintos/src/lib/user/syscall.c>

additional system call인 Fibonacci, max\_of\_four\_int 함수를 정의했고, max\_of\_four\_int의 경우에는 argument가 4개가 필요한 syscall4 asm code가 필요하므로 새로 정의했다.

</pintos/src/lib/syscall-nr.h>

Additional system call인 Fibonacci, max\_of\_four\_int를 위한 system call number(SYS\_FIBO, SYS\_MAX)를 정의했다.

</pintos/src/threads/thread.h>

Thread.h에 정의돼 있는 thread 구조체에 process\_wait에서 필요한 변수, 구조체들을 추가적으로 추가했다. Struct list child, struct list\_elem child\_elem, semaphore/////

또한 exit할 시의 status를 저장할 exit\_status도 선언했다.

</pintos/src/lib/thread.c>

Init\_thread함수에서 semaphore를 초기화하기 위해 sema\_init함수를 사용했다. 또한 thread를 저장할 list에 thread t->child로 초기화했고 running\_thread()->child, t->child\_elem을 list에 push\_back했다.

</pintos/src/userprog/process.c>

Process\_execute함수에서 공백을 기준으로 명령어 부분을 저장할 parse[50]배열을 선언했고 strtok\_r함수를 사용할 때 필요한 임시 저장 포인터 char \*te를 선언했다.

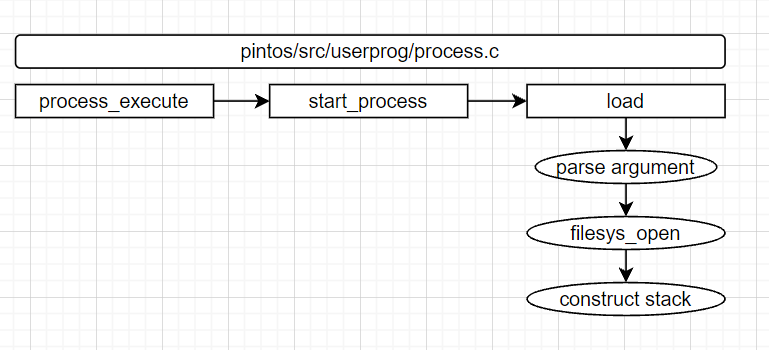
Process\_wait함수에서 thread를 저장할 list\_elem 구조체 포인터 list\_thread와 thread 구조체 포인터 temp, status 반환 값을 저장할 ret\_status를 선언했다. ////list와 관련된 함수들과 semaphore를 사용한 함수 sema\_up, sema\_down에 대한 자세한 설명은 4-B에서 한다.

load함수에서 argument의 개수를 저장할 integer변수 cnt와 공백을 기준으로 parse한 argument들을 저장할 2차원 char배열 parse[50][50]을 선언했다. 위와 마찬가지로 strtok\_r함수를 사용할 때 필요한 임시 저장 포인터 char \*te를 선언했다. Stack을 쌓을 때 추가한 변수로는 word alignment를 고려할 int 변수 align\_num과 argument가 쌓인 stack의 주소를 저장할 2차원 배열 address를 선언했다.

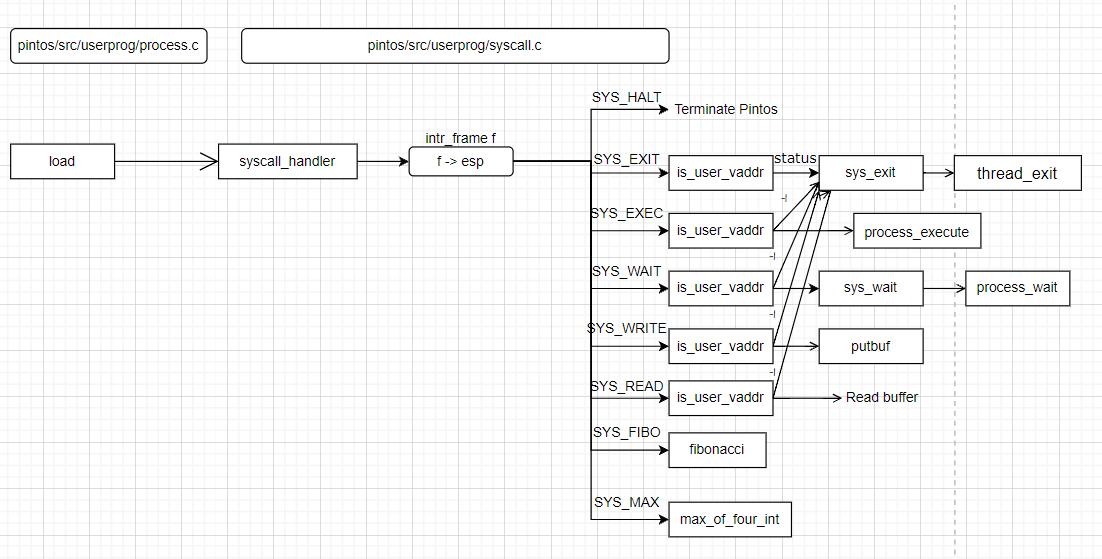
</pintos/src/userprog/syscall.c>

Syscall\_handler에서는 syscall number에 맞게 switch-case문을 구현했다. Argument가 있는 경우는 is\_user\_vaddr 함수를 이용해 적절한 공간을 사용하는지 확인했다. SYS\_EXIT의 경우 \*(uint32\_t\*)(f->esp+4) 즉 integer형인 status로 보내서 thread\_exit를 수행한다. SYS\_EXEC의 경우, process\_execute에 file\_name인 char 포인터 배열((const char\*)\*(uint32\_t\*)(f->esp+4))를 보내서 return받은 tid를 eax에 저장한다. SYS\_WAIT의 경우, 새로 정의한 sys\_wait함수로 thread id인 (tid\_t)\*(uint32\_t\*)(f->esp+4)를 보내서 그 값을 process\_wait함수로 보내서 return받은 값을 eax에 저장한다. SYS\_READ의 경우 buffer의 크기를 저장할 변수 read\_buf를 선언해서 \0아 나올때까지 읽고 read\_buf를 eax에 저장한다. SYS\_WRITE의 경우, file descriptor가 1인 경우 putbuf를 수행한다. Buf를 put한 것을 eax에 저장했다.

1. **연구 결과**
   1. **Flow Chart**
2. Argument Passing



1. User Memory Access / System Calls



* 1. **제작 내용**

1. Argument Passing

텍스트이(가) 표시된 사진

자동 생성된 설명

Argument를 입력 받아서 제대로 명령어를 수행하기 위해 앞의 명령어 부분과 argument 부분을 parse해서 pintos의 user virtual memory에 있는 user stack에 쌓는 작업을 수행해야 한다. /pintos/src/userprog 폴더에 있는 process.c 함수에서 load함수에서 parameter로 들어온 file\_name을 parsing한다. strtok\_r 함수를 사용하여 argument parsing을 수행하여 2차원 char 배열로 정의한 parse에 parsing한 argument들을 저장한다. filesys\_open 함수를 통해 executable header를 read한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

cnt는 parsing한 argument들의 개수 parse[cnt-1] 부터 parse[0] 까지를 stack에 쌓는 반복문의 코드이다. ‘\0’(null) 값을 포함한 parse argument의 길이를 stack pointer(\*esp)에서 감소시킨 후 그 부분의 stack pointer에 parse[i]값을 넣는다. 이 때 strlcpy 함수를 사용한다. 그리고 address[i]에 현재 stack pointer, 즉 주소를 저장한다. Word alignment를 고려하기 위해 변수 align\_num를 사용했는데 반복문의 수행시마다 ‘\0’(null) 값을 포함한 parse argument의 길이를 더하고 4byte 즉 4보다 align\_num이 크면 4보다 작거나 같을 때까지 -4를 수행한다. 4인 경우는 word alignment할 필요가 없기 때문에 4인 경우도 포함되었다. Argument의 값들을 모두 넣었으면 stack pointer에서 (4-align\_num)을 감소시켰다. 후에 -4를 추가적으로 해서 word alignment를 수행했다. 그 stack에는 0을 쌓았다.

텍스트, 화면, 은색이(가) 표시된 사진

자동 생성된 설명

Argument 개수만큼 반복문을 진행하면서 address[i]의 값들, 즉 parse[i]의 stack에 쌓인 주소를 stack에 순차적으로 쌓았다. Argv의 주소를 stack에 쌓고 argument의 길이인 cnt를 쌓고 최종적으로 return address인 0을 stack에 쌓았다. Stack에 쌓일 때는 4byte씩이므로 각각 \*esp에서 4씩 감소시켰다.

argument를 parsing하는 strok\_r, string을 복사해주는 함수 strlcpy(), string의 길이를 반환해주는 strlen함수는 pintos/src/lib에 정의되어 있다.

1. User Memory Access

pintos/src/threads/vaddr.h에 정의되어 있는 is\_user\_vaddr, is\_kernel\_vaddr을 사용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

is\_user\_vaddr은 현재 address가 PHYS\_BASE보다 작다면, 즉 user virtual address인지를 확인해서 맞다면 true를 반환해주는 함수로 syscall\_handler에서 system call을 수행하기 전에 (f->esp + 4) 접근하려는 주소가 virtual address가 아니라면 syscall.c에서 정의한 sys\_exit(-1)을 호출해서 thread를 exit한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Is\_kernel\_vaddr은 현재 address가 kernel virtual address에 있는지를 확인하는 함수이다. pintos/src/userprog/exception.c에서 “threads/vaddr.h”를 include시켜서 is\_kernel\_vaddr을 사용하게 했다. pintos/src/userprog/exception.c의 page\_fault함수에서 user가 0이거나 is\_kernel\_vaddr(fault\_addr)이 True, 즉 address가 kernel영역에 있는 경우 userprog/syscall.h에 있는 sys\_exit(-1)을 수행했다. 이를 위해 “userprog/syscall.h” 헤더파일도 include해주었다.

1. System Calls

pintos/src/lib/user/syscall-nr.h에서 enum type으로 저장된 system call number들을 사용한다. pintos/src/lib/user/syscall.c에 있는 halt, exit, exec, wait, read, write 함수에서 각각에 argument 개수에 따라 syscall1, syscall2, syscall3를 반환하고 각각에 맞는 system call number를 parameter로 같이 보낸다.

텍스트이(가) 표시된 사진

자동 생성된 설명게이지이(가) 표시된 사진

자동 생성된 설명

pintos/src/userprog/syscall.c에서 syscall\_handler 함수를 구현했다. Parameter로 넘어온 intr\_frame f에 대해서 \*(uint32\_t\*)(f->esp) 값으로 switch문을 수행해서 값에 맞는 부분을 수행한다. 이 값은 pintos/src/lib/syscall-nr.h에서 정의한 system call number이다.

* halt()

halt는 Pintos를 끝내는 system call이다. shutdown\_power\_off 함수를 사용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* exit()

현재 user program 즉, 현재 thread을 종료하고 kernel에 status를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

* exec()

텍스트이(가) 표시된 사진

자동 생성된 설명

pintos/src/userprog/process.c에 있는 process\_execute() 함수를 호출한다. f->eax에 process\_execute에 file\_name에 해당하는 (const char\*)\*(uint32\_t\*)(f->esp+4)를 넣어 보낸 후 return값을 넣는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Process.c에서 process\_execute 함수로 넘어온 file name은 전체 argument 이므로 filesys\_open 함수로 open할 명령어를 parse해서 넘겨준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

child thread를 create해서 return해준다. syscall\_handler에서는 f->eax에 return 받은 tid값을 저장한다.

* wait()

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

wait system call은 child process가 끝날 때까지 그 process를 기다리는 system call이다. child thread ID가 valid인지를 확인하면서 child thread가 죽었을 시 exit\_status를 child thread로부터 가져온다.

Pintos/src/userprog/syscall.c에서 sys\_wait함수를 정의해 syscall\_handler함수의 switch문에서 SYS\_WAIT인 경우 f->esp+4의 값을 (tid\_t)\*(uint32\_t\*)의 type으로 보내서 process\_wait함수에서 반환된 값을 반환해 f->eax값에 넣는 작업을 했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

pintos/src/threads/thread.h에서 thread 구조체 선언부분에 userprog에서 사용되는 변수구조체들을 새로 정의했다. Process\_wait에서 사용하기 위해 thread를 넣을 list 구조체 child를 정의하였고 list\_elem child\_elem을 정의했다. 또한 wait의 기능을 수행하기 위해 thread를 정확하게 사용하기 위한 semaphore p, p2를 정의했다. exit\_status는 child process를 exit할 시의 status를 저장하는 변수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Process\_wait함수를 살펴보면 child thread가 error가 아닌지를 확인한 후 list\_elem 구조체 포인터로 구현한 list\_thread에 list의 begin에 존재하는 현재 thread(thread\_current())의 child를 넣는다. 반복문을 진행해서 list의 마지막까지 도달하면 break문을 수행하며 제대로된 return 방식은 child의 thread id인 child\_tid와 돌고있는 temp->tid가 일치할 때 끝나는 thread의 exit\_status를 반환하는 것이다. 임시의 thread 구조체 thread포인터 temp에 list\_entry(list\_thread, struct thread, child\_elem)의 결과를 넣고 temp의 tid가 child\_tid와 같은 경우 &(temp->p)를 sema\_down 하고 temp의 exit\_status를 ret\_status 변수에 저장한 후 temp->child\_elem을 list에서 remove한 후 &(temp->p2)를 sema\_up한 후 ret\_status를 반환하고 child\_tid와 같은 것을 찾지 못하고 list의 끝까지 탐색하면 -1을 반환한다. Sema\_up과 sema\_down은 semaphore를 쓴 것으로 thread가 수행 중에 다른 thread로부터 interrupt를 받으면 코드의 수행에 문제가 생길 수 있기 때문에 lock을 걸어서 해당 동작이 interrupt없이 제대로 수행되도록 설정해준 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명 마지막으로 pintos/src/userprog/process.c의 process\_exit함수에서 process\_wait에서 sema\_up, sema\_down 시켰던 것을 각각 sema\_down, sema\_up을 진행하며 free시켜준다. 이때 cur은 현재 thread를 의미한다.

* Write()

텍스트이(가) 표시된 사진

자동 생성된 설명

Write 함수는 Project1에서는 간단히 STDOUT으로 write하게 구현했다.

(int)\*(uint32\_t\*)(f->esp + 4), 즉, file descriptor이 1(STDOUT)인 경우, putbuf함수로 사용하여 write함수를 구현했고 그 넣은 값을 eax값에 넣었다.

* Read()

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

read 함수는 Project1에서는 간단히 STDIN에서 read하도록 구현했다.

(int)\*(uint32\_t\*)(f->esp + 4), 즉, file descriptor이 0(STDIN)인 경우, (char\*)(void\*)(f->esp + 8)을 반복문에서 input\_getc함수를 통해 character 하나를 읽어 ‘\0’(null)값을 찾을 때 break한다. Read한 buffer의 길이를 eax값에 넣었다.

1. Additional System calls

fibonacci와 max\_of\_four\_int의 새로운 system call을 정의한다. pintos/src/lib/syscall-nr.h에 SYS\_FIBO, SYS\_MAX를 새로 정의했다.

텍스트, 실내, 어두운이(가) 표시된 사진

자동 생성된 설명

pintos/src/lib/user에서 max\_of\_four\_int의 경우는 argument가 4개이므로 새로 syscall4의 asm code를 정의했고 syscall 함수를 정의했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

pintos/src/examples에 additional.c 함수를 새로 정의했다.

텍스트이(가) 표시된 사진

자동 생성된 설명 함수 코드를 살펴보면 pintos/src/lib/stdlib.h에 정의 되어있는 atoi함수를 사용했다. atoi함수는 문자열을 integer 타입으로 변환시키는 함수로 argv[1 … 4]로 integer로 변환해서 맨 앞 정수인 n(argv[1])을 Fibonacci 함수로 보내서 피보나치 값을 반환 받고 argv[1…4]의 n, a, b, c를 max\_of\_four\_int로 보내서 가장 큰 수를 반환 받았다.

텍스트이(가) 표시된 사진

자동 생성된 설명 pintos/src/userprog/syscall.c 에서 fibonacci와 max\_of\_four\_int함수는 다음과 같이 정의했다.

* 1. **시험 및 평가 내용**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 병이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

21 pass (Project1)

* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**