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

담당 교수 : 김영재 교수님

조 / 조원 : 20211608 홍지연

개발 기간 : 2023.10.02 ~ 2023.10.08

1. **개발 목표**

유저모드에서 실행되는 유저 프로그램으로부터 Pintos OS 환경에 의해 system call handler와 system call 일부 기능이 제공되도록 한다. 현재 핀토스는 유저 프로그램을 부르고 실행하는 것은 가능하다. 하지만 입력된 명령어에 해당되는 유저 프로그램을 불러들이는 것과 유저 프로그램의 실행에 따라 발생하는 시스템 콜을 처리하지 못한다. 이를 위해 1) argument passing, 2) user memory access 3) system call을 구현한다. System call는 기본적으로 halt, exit, exec, wait, read, write와 추가적으로 피보나치 수열의 결과를 구하는 fibonacci와 4개의 수 중 최댓값을 구하는 max\_of\_four\_int를 구현한다.

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

터미널에서 입력 받은 전체 argument조합을 띄어쓰기를 기준으로 개별적으로 구별하고 stack memory에 high address에서 low address 순서로 쌓는다.

1. User Memory Access

사용자가 메모리에 접근하는 과정에서 올바르지 않은 주소로 인해 발생하는 오류를 방지하고자 예외 처리를 수행한다.

1. System Calls

요청된 system call에 알맞은 명령어를 system call handler를 통해 처리하여 수행한다. 이때 필요할 경우 stack register인 esp를 이용하여 stack에 접근해 저장된 argument를 사용한다.

* 1. **개발 내용**
* Argument Passing

인자 조합의 입력을 strtok\_r() 함수를 사용하여 띄어쓰기를 기준으로 인자들을 구분하고 argv 배열에 저장한다. (이때 strtok\_r()은 대상인 문자열에 영향을 주므로, strlcpy() 함수를 사용하여 복사한 fn\_copy를 사용한다.) 이 과정에서 인자들의 개수인 argc를 구할 수 있다. 다음으로 스택 포인터 esp를 사용하여 high address에서 low address 순서대로 인자들인 argv의 값, word align, null, argv의 주소, argc, return address(0)를 스택에 쌓는다.

* User Memory Access

Pintos에서 유저 프로그램이 프로세스로 등록될 때, 각 프로세스마다 4GB의 유저 가상 메모리가 생성된다. 여기서 0GB~3GB는 유저 영역이고, 나머지 3GB~4GB는 커널 영역에 매핑된다. 유저 프로그램이 유저 영역이 아닌 커널 영역의 메모리에 접근하는 것을 invalid memory access라고 한다.

* + Invalid memory access를 어떻게 막을 것인지 설명

Invalid memory access를 막기 위해 Pintos 상에서 해당 메모리 주소가 커널 영역이 아닌 유저 메모리를 접근하고 있는지, 즉 유저 가상 메모리 영역에서 0GB~3GB 범위를 접근하고 있는지 확인한다. 이를 확인하기 위해 threads/vaddr.c의 is\_user\_vaddr() 함수와 is\_kernel\_vaddr() 함수를 사용한다.

* System Calls

유저 모드에서 메모리의 커널 영역에 접근할 수 없다. 따라서 유저 프로그램이 유저 모드에서 실행될 때 OS가 제공하는 기능을 이용하는 것이 불가능하다. 유저 프로그램에서 OS가 제공하는 프로그램을 사용하기 위해 OS가 제공하는 시스템 콜에 대한 연결고리 역할을 수행하는 시스템 콜 인터페이스가 필요하다.

Halt: shutdown\_power\_off() 함수를 사용하여 핀토스를 종료시킨다.

Exit: 현재 실행 중인 유저 프로그램을 종료한다. 콘솔 창에 exit 메시지를 출력하고, exit status를 부모 프로세스에게 알리고, 해당 프로세스가 사용하던 모든 자원을 반환한다.

Exec: 자식 프로세스를 생성하고, 사용자 프로그램 file을 실행한다.

Wait: pid의 자식 thread id가 유효한지 확인하고 자식 thread가 끝날 때까지 부모 thread가 기다렸다가, 자식 thread가 종료되면 exit status를 받는다.

Read: 유저 프로그램으로부터 받은 입력을 STDIN으로 읽는다.

Write: 유저 프로그램으로부터 받은 입력을 STDOUT으로 콘솔 창에 띄운다.

Fibonacci: n번째 피보나치 수열의 결과를 반환한다.

Max\_of\_four\_int: 4개의 정수 a, b, c, d 중 최댓값을 반환한다.

유저 프로그램에서 OS에 시스템 콜을 요청한다. lib/user/syscall.c에서 시스템 콜에 대해 정의된 함수에 따라 유저 모드에서 해당 시스템 콜의 넘버, 인자들, interrupt frame을 레지스터에 채우고 lib/syscall-nr.h에 저장된 시스템 호출 번호를 CPU 레지스터에 저장한다. Syscall 명령을 CPU에게 내리면 소프트웨어 interrupt 발생으로 인해 CPU가 커널 모드로 전환된다. system call handler가 userprog/syscall.c의 syscall\_handler() 함수를 통해 시스템 콜에 의한 interrupt, 즉 시스템 콜의 요구사항을 수행하고, interrupt frame의 eax 변수에 시스템 콜을 처리한 값을 return하여 반환 값을 저장한다. 모든 시스템 콜을 처리한 이후, 유저 스택에 쌓아두었던 esp 포인터를 복원하여 유저 모드로 다시 돌아가고, 시스템 콜의 결과는 eax에 저장된 반환 값을 통해 알 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

2023.10.02 ~ 2023.10.04: argument passing implementation.

2023.10.05 ~ 2023.10.07: user memory access, system call implementation.

2023.10.07: additional implementation.

2023.10.08: 보고서 작성

* 1. **개발 방법**

userprog/process.c에서 argument passing을 구현한다. 먼저 process\_execute에서 argv[0]이 명령어에 대한 동작이 담긴 파일 이름으로 전달되도록 parse file name을 수행한다. File\_name을 strlcpy()와 strtok\_r() 함수를 이용하여 띄어쓰기를 기준으로 첫 번째 인자를 분리한 뒤, file을 생성하는 filesys\_open()의 인자로 넣어준다. 다음으로 load에서 file\_name을 strlcpy()와 strtok\_r() 함수를 이용하여 띄어쓰기 단위로 모든 인자들을 구별한 뒤 각각 argv 배열에 저장하고, file을 여는 함수들에 argv[0]을 인자로 사용한다. 그 다음, setup\_stack()이 수행된 후 stack pointer esp를 통해 stack memory에 인자들을 쌓는다.

Userprog/syscall.c. userprog/syscall.c에서 system call이 호출하는 함수들을 구현한다. 각 시스템 콜은 switch문을 통해 각 명령어에 대한 함수를 호출하여 수행하는데, exit, exec, wait는 먼저 syscall.c의 syscall\_handler에서 is\_user\_vaddr()을 사용하여 인자가 유효한 유저 메모리를 참조하는지 user memory access를 검사한다. 인자가 유저 메모리 밖을 참조하는 경우는 userprog/exception.c의 page\_falut()에서 is\_kernel\_vaddr() 함수로 검사하도록 추가 구현한다. 다음으로 각 시스템 콜에 대한 함수를 구현하고 userprog/syscall.h에 모든 함수들을 선언하는데, exec와 wait의 경우 userprog/process.c의 함수를 사용하므로 syscall.h에 userprog/process.h를 include하고, exec에 대한 함수 sys\_exec이 pit\_t 자료형을 반환할 수 있도록 pid\_t를 선언한다.

System call의 wait을 구현하기 위해 userprog/process.c의 process\_wait() 함수를 구현한다. 이를 위해 userprog/thread.h의 thread에 멤버들을 추가하고, 이를 위해 “synch.h”와 “vaddr.h”를 include한다. 또한 Userprog/thread.c의 init\_thread()에서 해당 멤버들을 thread에 초기화하는 동작을 추가 구현한다. 그리고나서 child thread의 일이 끝날 때까지 부모 thread가 기다리는 과정을 process\_wait()에서 list\_begin(), list\_end(), list\_entry() 그리고 sema\_down(), list\_remove(), sema\_up을 이용하여 구현하고, 마지막으로 process\_exit()에 child\_sema를 sema\_up(), memory\_sema를 sema\_down한다.

마지막으로 Fibonacci와 max\_of\_four\_int는 lib/user/syscall.h, lib/user/syscall.c, lib/syscall\_nr.h에서 추가 구현한다. Syscall\_nr.h의 enum에 fibonacci와 max\_of\_four\_int에 대한 멤버를 추가하고, 4개의 인자를 받아야 하는 max\_of\_four\_int를 위해 syscall.c에 arg3까지 인자로 받는 syscall4를 추가한다. Fibonacci와 max\_of\_four\_int에 각각 syscall1과 syscall4를 사용한 함수를 syscall.c에 구현하고, 해당 함수를 syscall.h에 선언한다. 그리고 이렇게 구현한 시스템 콜에 대한 함수를 userprog/syscall.c와 userprog/syscall.h에 구체적으로 구현한다.

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

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

1. System Calls

텍스트, 도표, 평면도, 평행이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**

1. Argument Passing

* Userprog/process.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Process\_execute()에 들어온 file\_name은 인자들의 조합으로 구성된 문자열이므로, filesys\_open에는 argv[0]에 해당하는 명령어만 분류하여 인자로 넣어야 한다. 이를 위해 strlcpy()로 file\_name을 복사한 fn\_copy\_2를 띄어쓰기를 기준으로 strtok\_r()를 한 번만 사용하여 argv[0]을 ptr에 저장하고 이를 filesys\_open()의 인자로 넣어서 명령어를 파일명으로 인식하여 file을 생성하도록 만든다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

다음으로 load()에서 동일한 작업을 수행하되, argv에 모든 인자들을 저장하기 위해 argv[agrc]가 NULL이 될 때까지, 즉 모든 인자를 분리하여 strtok\_r()이 NULL을 반환할 때까지 strtok\_r()을 사용하여 인자를 분리하는 동시에 argc의 값을 계산한다. 다음으로 수행되는 파일 생성 프로세스에서 file\_name을 인자로 사용하던 모든 함수들의 인자를 명령어에 해당하는 argc[0]로 바꿔준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

다음으로 setup\_stack(esp)를 수행한 직후에 스택에 인자들을 쌓는다. 초기화된 스택 포인터 esp는 현재 high address를 가리키고 있고, stack을 쌓으며 esp가 점점 low address를 가리키게 된다. 먼저 argv[][…], 즉 argv[]의 값을 push하며 esp가 char\*의 크기만큼 감소하게 된다. 감소된 총 크기가 4의 배수인지 알 수 없으므로 word align을 수행해 포인터의 위치를 맞추고, 다음으로 차례대로 argv[], 즉 argv 원소들의 주소값과 argv 시작 주소, argc, return address를 push한다.

1. User Memory Access

* Userprog/process.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Exec에 대한 system call은 process.c의 process\_execute()를 사용한다. 따라서 process\_execute 내에서 thread\_create() 전에 file\_name에 대한 user memory access 검사를 추가하였다.

* Userprog/syscall.c

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Exit, exec, wait에 대한 시스템 콜에 대해 요구 조건을 수행하기 전, 메모리 접근이 유효한지 검사하기 위해 is\_user\_vaddr()를 사용한다. Switch문에서 사용을 간소화하기 위해 check\_address ()함수를 새로 만들어 진행하였다. Is\_user\_vaddr(vaddr)이 유저 메모리 영역이 아니라면 sys\_exit(-1)을 호출한다. Sys\_exit()은 system call 구현에서 구현할 예정이다.

* Userprog/syscall.h



구현한 함수의 prototype을 헤더 파일에 선언하였다.

* Userprog/exception.c

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

인자가 유저 메모리 밖을 참조하는 경우를 검사하기 위해 page\_falut()에서 검사하도록 user가 선언된 다음 자리에 구현하였다. 만약 user가 0이라면 sys\_exit(-1)을 호출한다. 또한 is\_kernel\_vaddr() 함수로 fault\_addr이 커널 영역인지 확인하고, 만약 그렇다면 sys\_exit(-1)을 호출한다.

1. System Calls

* Userprog/syscall.c

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

먼저 esp를 사용하는 각 인자를 각각 \*(uint32\_t)\* 자료형으로 변환하여 esp4, esp8, esp12, esp16으로 선언하였다. 다음으로 syscall number에 해당하는 \*(uint32\_t)\*(f->esp)에 따라 switch문으로 각 시스템 콜을 호출한다. 이때 SYS\_EXIT, SYS\_EXEC, SYS\_WAIT의 경우 check\_address()를 통해 user memory access 검사를 선행한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Sys\_halt는 shutdown\_power\_off 함수를 호출하여 처리한다.

Sys\_exit은 메시지를 출력한 후, 현재 실행 중인 thread의 exit\_status를 변경하고 tread\_exit을 호출하였다.

Sys\_exec는 process\_execute 함수에 인자로 file을 넣은 값을 반환한다.

Sys\_wait은 process\_wait 함수에 pid를 인자로 넣은 값을 반환한다.

Sys\_read는 fd가 0일 때 input\_getc() 함수를 이용하여 STDIN 출력을 수행한다.

Sys\_write는 fd가 1일 때 putbuf() 함수를 사용하여 STDOUT을 출력한다.

* Userprog/syscall.h

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이렇게 구현한 함수들의 prototype을 모두 헤더 파일에 추가하였다. 더불어, sys\_exec에서 사용하는 자료형 pit\_t을 추가하였다. 또한, exec과 wait를 제대로 구현하였음에도 관련 테스트에 실패하는 이슈가 있었는데, 뒤늦게 이슈를 해결하기 위해 sys\_exec와 sys\_wait에서 user/process.c의 함수를 사용하므로 “userprog/process.h”를 include하자 테스트에 성공하였다.

* Threads/thread.h

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Wait는 부모 프로세스가 자식 프로세스 종료까지 기다리는 기능이다. 그러므로 프로세스의 부모 자식을 구분할 수 있도록 thread 구조체에 멤버를 추가한다. List child는 프로세스의 child thread list를, semaphore child\_sema와 memory sema는 각각 child의 상태에 대한 lock, memory에 대한 lock을 수행하는 semaphore 구조체이고 list\_elem child\_elem은 child list의 element 그리고 int exit\_status는 exit 호출 시 종료 상태에 대한 변수이다. 이러한 멤버 추가를 위해 “synch.h”와 “vaddr.h”를 include하였다.

* Threads/thread.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

헤더 파일에 멤버를 추가한 직후 make check을 실행하였을 때, kernel panic이 발생하였다. 해당 이슈를 해결하기 위해 헤더 파일의 thread 구조체에 추가한 멤버들을 init\_thread()에서 초기화하는 부분을 추가하였다. Semaphore 변수에는 sema\_init(), list 변수에는 list\_init()과 list\_push\_back()을 사용하였다.

* Userprog/process.c

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

wait에서는 인자로 주어진 tid 프로세스가 끝날 때까지 기다려야 한다. 자식 프로세스가 실행될 때 sema\_down()으로 부모 프로세스를 대기시키고, 자식 프로세스가 종료될 때 sema\_up()으로 부모 프로세스를 재개한다. Memory lock을 위해 memory\_sema 또한 sema\_up()으로 list\_remove()로 제거하기 전까지 보존한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

process\_exit에서는 child thread가 종료될 때 semaphore를 풀어주기 위해 process\_wait와 반대로 child\_sema에 대해 sema\_up()을 하고 memory\_sema에 대해 sema\_down()을 수행한다.

1. Additional System calls

* Lib/syscall\_nr.h

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Enum에 fibonacci와 max\_of\_four\_int에 대한 system call number를 추가한다.

* Lib/user/syscall.c

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Max\_of\_four\_int()에는 총 네 개의 인자가 들어가므로, 이를 위해 arg3까지 인자로 사용하는 syscall4를 추가한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음으로 fibonacci와 max\_of\_four\_int를 각각 인자를 한 개 사용하는 syscall1과 인자를 네 개 사용하는 syscall4에 system call number와 인자들을 넣어 호출한다.

* Lib/user/syscall.h

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

Syscall.c에서 구현한 함수의 prototype을 헤더 파일에 추가한다.

* Userprog/syscall.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

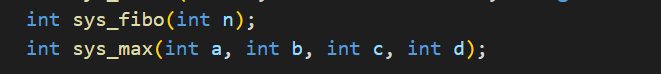
추가로 구현한 시스템 콜을 switch문에 추가한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

더불어 해당 시스템 콜의 요구 조건을 수행하는 함수를 구현한다.

* Userprog/syscall.h



추가 구현한 함수에 대한 prototype을 헤더 파일에 추가한다.

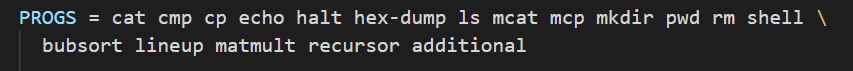
* Examples/additional.c

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

시스템 콜을 제대로 구현하였는지 확인하는 테스트 코드를 작성하였다.

* Examples/makefile





추가한 테스트 코드가 컴파일되도록 makefile에 추가한다.

* 1. **시험 및 평가 내용**

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명