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

담당 교수 : 문의현

학번 / 이름 : 20171300 / 지용환

개발 기간 : 2022/10/5-2020/10/29

1. **개발 목표**

핀토스의 system call을 구현하는 것을 주된 목표로 한다. 이 system call을 구현하기 위해서는 명령을 인식하게 하는 스택 즉 argument passing, 메모리 보호를 위한 user memory access를 구현한 이후, 여러가지 system call (halt, exit 등등)을 구현을 해야한다. 추가적으로 Fibonacci, max of four int또한 구현한다.

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

명령어를 스택에 저장시키는 것으로, 각 명령어의 인자(예를들어 echo x시엔 echo, x)들, 그 인자의 주소 등을 인식하기 쉽도록 스택에 정리해서 저장시킨다.

1. User Memory Access

Memory에 접근할 때, 잘못된 주소가 넘겨질때의 오류처리를 위한 것으로, 핀토스에 정의된 함수를 이용해 종료시킨다..

1. System Calls

System call handler를 통해, 1번의 argument passing을 통해 스택에 쌓은 명령어를 읽으며, 상황에 맞게 처리시킨다.

* 1. **개발 내용**
* Argument Passing

먼저 parsing을 해야하는데 간단히 예를 들면 echo x가 들어왔을 시 echo 와 x로 나누어 주는 것이다. 이는 띄어쓰기로 나눌수 있으므로, strtok\_r로 나누어 argv에 저장을 했다. 이후에 make\_stack을 통해 argument passing을 진행하였는데, 먼저 argv, 이후에 word align(stack의 주소를 4의 배수로 맞춰주기 위함. 이 칸부터는 stack에서 4칸씩만 이동하면 다른 데이터가 나오는 것을 확정시킬수 있다), 0, 다음으로 argv 데이터의 주소, argv의 주소, 데이터 개수,0을 스택에 차례대로 저장시킴으로써 argument passing이 종료된다. 이때 스택은 esp로써, make\_stack의 real parameter중 하나이다.

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

Invalid memory access란 말 그대로 잘못된 메모리의 접근으로, user process가 user memory의 공간이 아닌, kernel memory에 접근한 상황이다. 핀토스에서는 user process가 등록되면, 4GB의 가상메모리가 생기고 3GB를 user가 접근 가능한 physical baseline, 즉 user영역이고 3GB~4GB영역은 kernel의 영역이다. 이 상황에서 user process가 kernel의 영역에 접근하는 것을 막아야 여러 에러를 방지할수 있다.

* + Invalid memory access를 어떻게 막을 것인지 설명

앞서 설명했듯이, physical baseline보다 아래에 있는지를 확인하면 된다. 이를 확인하는 함수는 핀토스에서 제공하고 있고, is\_user\_vaddr, is\_kernel\_vaddr이다. 각각 유저영억인지, 커널영역인지, true, false를 반환해주는 함수이다. 즉 조건문으로 ! is\_user\_vaddr을 이용해 user메모리가 아니라면 참을 반환시켜(!때문에 참) invalid memory access임을 확인하고 exit시키면 잘못된 접근을 방지할 수 있다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

위에서 설명했듯이, 핀토스는 user와 kernel모드 로 나누어져 있는데, 유저 프로그램이 실행되는 상황에서는 memory access때문에 kernel영역에 접근하는 것이 불가능하다. 하지만 os가 제공하는 읽기,쓰기 등의 많은 기능은 kernel영역에 접근을 해야만 하므로, user 모드에서 kernel 모드의 기능을 사용하게 하기 위해서 시스템 콜이 필요한 것이다.

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

halt: 핀토스를 종료시킨다.

exit: 현재 실행중인 user프로그램을 종료하고, exit status를 부모 process에 반환시켜준다. Exit status가 0이면 정상 종료이고, 아니라면 비정상 종료를 의미한다.

exec: 새로운 자식 프로세스를 생성후에, 그 프로세스를 실행시키는 함수이다. 이때 새로운 프로세스 id를 반환시킨다.

wait: 자식 프로세스의 작업에 종료될때까지 기다리게 시키는 시스템 콜으로써, 먼저 pid가 유효한 자식 thread인지 확인 후에, 자식이 종료되면 exit status를 가져온다.

read: 사용자 프로그램으로부터 받은 내용을 읽고, 읽은 byte를 return한다. 현재의 프로젝트에선 STDIN으로부터만 받앗다.

write: 사용자 프로그램으로부터 받은 내용을 wite하고, 쓴 byte를 return한다. 현재는 STDOUT으로만 write했다.

fibonacci: n번째 피보나치 수열의 결과를 반환시켜준다.

max\_of\_four\_int: 4개의 정수를 입력받고, 그중 최대값을 반환시켜준다.

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

가장먼저 src/examples의 유저 레벨 프로그램에서 필요한 작업 함수를 호출하는데, 모든 함수는 src/lib/user/syscall.c에 정의가 되어있어야 한다. 이 파일에서 부합하는 함수를 찾아, system call 번호와 argument를 스택에 push시키고 interrupt를 걸어 system call의 발생을 알려준다. interrupt는 src/threads/interrupt.c의 interrupt handler가 수행되도록 하여, interrupt service routine, 즉 system call handler의 단계로 넘어가게 된다. src/userprog/syscall.c에서 system call handler()에 의해 system call을 수행하게 되는데, argument passing을 한 esp를 통해 system call의 내용을 확인한다. system call 처리 이후에는 처리 결과를 eax에 저장시킴으로써 사용자가 결과를 볼수 있도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

10/8 ~ 10/9 : argument passing, user memory access

10/10 ~ 10/17 : system call 구현

10/26 ~ 10/27 : 추가구현

10/28~10/30: 보고서 작성

* 1. **개발 방법**

1. **argument passing**

src/userprog/process.c와 src/userprog/process.h에서 구현하는 것으로 bool load란 함수 내에서 모든 작업을 하면된다. load함수의 dummy parameter엔 명령어를 의미하는 (echo x)file\_name이 있는데, 이를 전달받아 각각 echo와 x로 파싱시켜주었다. 이때 나눈 기준은 띄어쓰기로 간단히 strtok\_r을 이용하면 나눌수 있다. 스택은 setup\_stack이 끝난 이후, make\_stack으로 만들어 주었으며 esp를 이용해 스택(스택 포인터) 을 차례대로 순서에 맞춰 쌓아 주면 된다.

1. user memory access

유저 프로그램에서 잘못된 메모리 영역을 참조했을 때를 방지하기 위해 시스템 콜을 하기 전에 모든 argument들이 유효한 유저 메모리 안을 참조하는지 점검하는 함수를 구현하면 된다. 즉, src/userprog/syscall.c에 유저 메모리 영역인지 확인하는 user\_memory\_access란 함수를 선언한 이후, syscall\_handler에서 각각의 시스템콜 호출하기 전에 우선적으로 user\_memory\_access를 호출시킨다.

또한 src/userprog/exception.c에서 page\_fault란 함수가 메모리 관련 오류상황을 판단하므로 유저 영역 밖을 참조하는 경우 (user==0, 혹은 fault\_addr이 커널영역)에 exit를 시켜주면 된다.

3. system call

src/userprog/syscall.c (syscall.h)에 구현을 하는 함수로 본 프로젝트의 주된 목적이다. 먼저 src/lib/syscall-nr.h란 파일에서 선언된 system call의 종류들을 switch 구문으로 나누어 각 케이스에 맞는 system call을 수행하게 해야한다. 즉 새로운 system call 구현을 위해선 syscall-nr.h에 추가하는 것이 필수이다. 이때 system call의 종류와 인자는 src/lib/user/syscall.c에 정의가 되어있어, 이를 바탕으로 그대로 src/userprog/syscall.c에 정의했고, 메모리 검사 이후, esp, 즉 1번의 argument passing을 통해 만든 스택을 인자로 넘겨받게 하였다. 몇몇 return value를 필효로 하는 함수는 eax에 정의를 한다면, 사용자가 이를 읽을 수 있게 된다.

대부분의 system call은 단순히 그냥 핀토스에 미리 정의된 것을 거의 따라 갔지만, 몇몇 추가적으로 구현을 필요로 하는 함수가 있었는데 wait 구현을 위해선 synchronize가 필요했기에, src/thread/threads.h에 USERPROG에서만 이용하는 구조체를 구현했다. (#ifdef이용) 추가한 내용은 기다리는 것을 위한 세마포어, 자식의 관리하기 위한 저장을 리스트(\_list\_elem), 그리고 종료 상태를 의미한 exit\_c(이것은 exit 시스템콜에서도 이용)이다. src/thread/thread.c의 init\_thread에서 구조체를 초기화 해주는 코드를 추가한 이후, 이를 이용해 src/userprog/process.c의 process\_wait부분에서 세마포어를 이용, child와 parent의 대기 관계, 반환 값 등을 처리하며 구현하면 된다. process\_exit부분에서도 마찬가지로 세마포어를 이용하면 된다.

4. 추가 함수(Fibonacci, max\_of\_int\_four)

위에서 간단히 설명했듯이, syscall-nr.h에 정의한 이후, src/lib/user/syscall.c에 인자가 4개이므로 syscall4를 추가적으로 구현해 함수의 인자값 정보를 추가해 주면된다. 또한 src/userprog/syscall.c에선 관련 함수 처리 내용을 만든 후, src/examples에 addiontional.c를 통해, 출력문을 만들어 주면 된다.

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

주된 부분은 make\_stack이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

주된 부분은 user\_memory\_access이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. System Calls

주된 부분은 system call의 종류로서 각 system call을 사용한 이후, 다시 어떤 함수를 이용해 처리되는지까지 표현한 flow차트이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각 함수의 내용은 B에서 자세히 설명.

* 1. **제작 내용**

**가장 유의할 점은 반드시 wait을 구현해야지 결과가 보인다는 것이다.(이게 없으면 부모 스레드가 자식을 기다리지도 않고 그냥 종료).**

1. Argument Passing

src/userprog/process.c

먼저 parsing관련해선 단순히 file\_name을 공백 기준으로 나누어, argv에 저장하게 하였다. command는 argument의 개수다. 또한 나눈 것의 첫번째 부분(실질적인 command)을 file\_command로 따로 정의해 (예를들어 echo x라면 echo)이 부분을 새로운 file\_name으로 이용한다. 즉 가장 아랫줄의 filesys\_open(file\_command)같은 부분이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

실질적인 argument passing은 setup\_stack이후 부분에서 make\_stack으로 따로 정의하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

간단한 변수를 설명하자면 argv는 parsing으로 나온 결과이고, command는 argv인자의 개수다. data\_len은 esp의 주소를 맞춰주기 위한변수이고, address는 argv주솟값을 저장하기 위함이다.

첫번째 for문은 esp에 argv데이터를 쌓으며 그 주소를 address에 저장하고, 사용한 비트를 누적한다(data\_len사용). argv를 다 쌓았다면, data\_len을 계산해 4의 배수가 아니라면, 4의 배수 부분을 가르키게 하고, 이후 null, address, argv(그냥 윗칸임),. command,0을 차례대로 쌓는다. 이부분에서 실수로 word\_align을 계산하지 않고 스택을 쌓다가 정확한 명령이 전달되지 않는 오류가 있었었다. hex\_dump를 사용하면 된다고 했지만, 아무리 봐도 읽는 법을 몰랐기에, 본인처럼 모르겠다면 그냥 모두 %p를 찍으며 확인해보면 된다. 개개인마다 위치의 차이가 있을수도 있겠지만, 잘 저장이 된것이라면 아래와 같이 나오면 된다.

주소들이 4씩 차이가나고, 가르키는 (go)가 제대로 된 주소를 가르키는지 쉽게 확인이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. User memory access

src/userprog/syscall.c

아주 단순히, 그냥 user영역이 아니면 exit를 시키게 만들면 되는 함수이다. 이때 is\_user\_vaddr이란 함수를 이용하며 이 함수자체가 user영역인지 확인시켜주는 핀토스에서 미리 정의된 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

사용은 다음예시와 같이 일일히, 먼저 vaild한 접근인지 확인후, system call을 하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

src/userprog/exception.c

프로젝트 자료에 나와있듯이, page\_fault에서도 유효성을 검사해야하고, 이경우, user==0이거나, fault\_addr이 kernel영역에 있으면 invaild이므로 이 두가지 경우에만 exit하도록 추가해주면 된다. 여기서 user는 user메모리 접근이면 true, kenel접근이면 false(0)인 변수이고, fault\_addr은 page\_fault가 발생한 가상 주소로서, 이또한 kernel이면 invaild인 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. System Calls

**scr/userprog/syscall.c**

먼저 전체적인 syscall\_handler부분이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각 case의 부분은 src/lib/syscall-nr.h에 정의가 되어있고, 이에 따라 먼저 memory 검사후, src/lib/user/syscall.c에 미리 정의된 함수들을 그대로 syscall.c에서 구현하면된다. 각각의 구현 내용은 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

halt는 미리 정의된 shutdown\_power\_off를 이용한다. 이 함수는 미리 정의된 종료함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

exit는 exit출력 이후, 현재 스레드의 exit\_c(status)를 변경해준이후, thread\_exit를 이용해 처리했다. thread\_current()는 현재 실행중인 스레드의 구조체를 반환하는 것이고, thread\_exit는 현재 실행중인 스레드를 종료하는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

exec은 process\_execute를 실행후, 실행결과를 반환함으로써 처리했고, process\_execute의 변경점은 나중에 설명하겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

wait은 process\_wait를 실행후, 실행결과를 반환함으로써 처리했고, process\_wait의 변경점은 나중에 설명하겠다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

read함수는 input\_getc로 fd가 0이면(즉 stdin이면) 읽게 해주었다. input\_getc는 입력버퍼에서 문자를 retrieves해 반환하는 함수이다. 반환값은 제대로 읽었다면 사이즈, 그외 -1을 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

wirte는 fd=1이면(stdout)현재 버퍼의 사이즈만큼의 문자를 출력시켜주는 putbuf함수를 이용했고, 반환값은 역시 size이다.

src/userprog/process.c process\_execute

계속 execute가 실패해 뭐가 원인인지 몰랐었지만 답은 간단했다. 단순히 load때처럼 parsing을 해주면 되는 것이었다. 정확히 말하자면 parsing까지는 필요없고, 그냥 명령어만 제대로 읽을수 있게, file\_name의 앞부분을 따로 받으면 되었다. load에서 이미 해서 충분한 것이 아니라, file\_name은 그대로 있기 때문에 새롭게 parsing을 해주어야 한다는 것을 까먹었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

추가적으로 만약 file이 null이라면 이또한 오류이므로 file을 닫고, -1을 return해주도록 하였다.

process\_wait를 설명하기 전에 미리 wait에 필요해서 추가적으로 정의한 구조체와 코드를 설명하겠다.

src/threads/threads.h

텍스트이(가) 표시된 사진

자동 생성된 설명

스레드 구조체에 추가한 항목들이다. exit\_c는 종료 status이고, child thread에 관한 항목을 위해 list,list\_elem, 그리고 wait을 위한 세마포어 s1, memory lock을 위한 세마포어 s2이다.

추가적으로 threads.c엔

텍스트이(가) 표시된 사진

자동 생성된 설명

의 코드를 추가해 세마포어 초기화와 리스트 초기화를 진행하였다.

process\_wait

src/userprog/process.c

텍스트이(가) 표시된 사진

자동 생성된 설명

주된 목적은 부모 thread가 자식 thread가 끝날 때 까지 기다리고, 끝나면 자식 thread의 exit\_c를 반환받으면 된다. 이때 자식 thread는 함수의 인자인 child\_tid이다. list 관련 함수 list\_begitn, list\_next, list\_entry 등에 관해서는 전 프로젝트에서 이미 했기에 설명을 생략하겠다. 현재 스레드의 자식의 리스트를 for문으로 차례대로 검사를 하며, 만약 child\_tid와 현재 검사중인 리스트가 같을 때, 그때 실행 및 기다리는 시점으로 s1을 다운시킴으로써, child를 기다리게 시킨다. 이때 중요한 점은 메모리가 lock이 되어야 하므로 s2는 sema\_up시켜 남기게 시킨다. child 스레드가 종료가 되었을 때, exit\_c를 반환시켜주면 부모 스레드가 받으며 wait가 종료되게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

당연히 wait을 한다면 다시 세마포어를 원상 복구 시켜주어야 한다. 즉, process\_exit부분 마지막에, child thread가 종료된다면, 현재 스레드의 s1을 up시킴으로서, 기다릴 필요가 없게, 그리고 s2를 down시킴으로서, 메모리 lock을 해제해주면 된다.

1. Additional System calls

먼저 src/lib/syscall\_nr.h에 추가적인 함수를 정의해야 한다. (SYS\_FIBo…..)

텍스트이(가) 표시된 사진

자동 생성된 설명

다음으로 src/lib/user/syscall.c에 추가적 함수의 argument정보를 가진 함수를 추가한다. 새롭게 추가할 함수(MOF)는 인자가 4개이므로 다음과 같은 함수를 먼저 추가한 이후, 텍스트이(가) 표시된 사진

자동 생성된 설명

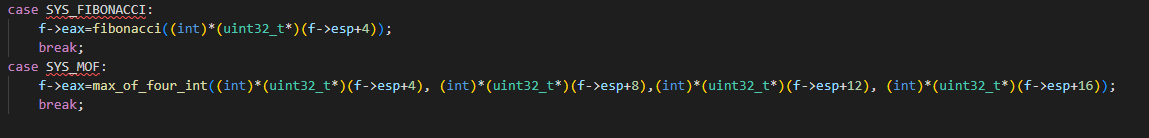
다음과 같게 인자를 받도록 만들어주면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이젠 system call을 구현한 것처럼 handler, 그리고 함수의 내용을 작성하면 된다.

src/userprog/syscall.c



피보나치 함수의 내용은 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그냥 일반적인 피보나치 구하는 함수이고, 크게 설명하지는 않겠다.

최댓값 구하는 함수의 내용은 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

단순히 if문으로 큰 것을 구한것이다.

마지막으로 이 함수들의 내용을 user가 읽을수 있도록, src/exampleds에 additional.c를 정의해 이 함수를 실행하도록 시켜준다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

examples/makefile엔 다른 명령어가 추가된 것처럼 같은 방식으로 추가시켜주면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. **시험 및 평가 내용**

pintos --filesys-size=2 -p ../examples/additional -a additional -- -f -q run 'additional 10 20 62 40'

를 입력했고 결과는 다음과 같다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**