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

담당 교수 : 김영재 교수님

조 / 조원 : 김동건(20211507)

개발 기간 : 2024.09.13 ~ 2024.09.22

1. **개발 목표**

주어진 pintos에서 구현되어 있지 않은 부분을 구현하여 목표로 하는 테스트 케이스들이 정상적으로 실행되게 하는 것이다. 이 과정에서 이번 프로젝트에서는 argument passing, user memory access, system calls를 구현한다. 또, 두 개의 추가 함수를 구현한다.

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

입력받은 명령어가 80x86 calling convention에 따라 parsing하여 stack에 쌓이게 된다.

1. User Memory Access

메모리 접근이 정상적인지 판단하고, 비정상적인 메모리 접근이 생길 때 pintos가 적절히 종료된다.

1. System Calls

6개의 system call(halt, exit, exec, wait, read, write)들이 system call handler를 통해 처리된다.

* 1. **개발 내용**
* Argument Passing
  + 커널 내 스택에 argument를 쌓는 과정 설명

process.c에 있는 load함수에서 파일 이름을 공백을 기준으로 분리한다. 분리된 정보들을 esp 포인터를 아래로 옮겨가며 쌓는다. 이때, 분리하면서 기록한 argument의 개수를 활용한다. 역순으로 쌓은 다음엔 4의 배수에 맞게 word-align을 해준다. 그 후, 쌓으면서 저장했던 주소들, argument 개수, return 주소를 마저 스택에 쌓아준다.

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

Pintos에서 invalid memory access는 유저 program이 NULL 포인터나, Unmapped virtual memory, 커널 영역을 전달하여 그 곳에 접근하는 것을 말한다.

* + Invalid memory access를 어떻게 막을 것인지 설명

주어진 pintos에 정의되어 있는 is\_user\_vaddr, is\_kernel\_vaddr, pagedir\_get\_page 함수를 이용하여 invalid memory access가 일어나는지 판단하고, 그렇다고 판단되면 exit(-1)을 하여 비정상적인 것을 알리며 프로그램을 종료시킬 것이다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

시스템 콜은 유저 영역에서 요청하는 기능들을 수행하게 하는 것이다. 이것이 필요한 이유는 user 영역과 kernel 영역은 서로 접근할 수 없기 때문이다. 즉, 시스템 콜은 이를 해결하기 위한 일종의 인터페이스라고 생각하면 된다.

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

1. halt: pintos를 shutdown\_power\_off()함수를 이용하여 종료시킨다.
2. exit: 현재 돌아가고 있는 user program을 종료하고 종료 status를 커널에 보고한다.
3. exec: 주어진 process를 실행시킨다.
4. wait: 주어진 부모 process가 자식 process가 완료되는 것을 기다린다.
5. read: 이번 프로젝트에서는 file descriptor가 stdin일 때, 명령어를 읽어 buffer에 저장한다.
6. write: 이번 프로젝트에서는 stdout를 putbuf로 수행한다.
7. fibonacci: n번째 피보나치 수를 리턴한다.
8. max\_of\_four\_int: 주어진 4개의 int값 중 가장 큰 값을 리턴한다.
   * 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

우선, 유저 레벨에서 시스템 콜 API 를 호출하면 interrupt가 발생되어 context switch가 일어나 커널 모드로 바뀌게 된다. 그 후, 시스템 콜 번호가 시스템 콜 테이블에 있는지 확인하는 과정이 진행된다. 만약 있다면 그에 맞는 작업이 진행된 후 다시 context switch를 통해 유저 레벨로 돌아오게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

09/13 ~ 09/15 강의 자료, 프로젝트 명세서, pintos 매뉴얼 공부

09/16 ~ 09/18 argument passing 구현

09/19 ~ 09/21 system call 구현 및 fail case들 디버깅

09/21 ~ 09/22 fail case 디버깅 및 수정, additional system call 함수 구현

09/22 ~ 09/23 보고서 작성

* 1. **개발 방법**

우선, argument passing은 userprog/process.c에 있는 load함수에서 구현하였다. 반복문을 통해 공백을 기준으로 주어진 명령어를 parsing하였다. 그 중 첫 단어는 real\_file\_name에 담아 파일을 열 수 있도록 처리해 주었다. 이는 실행할 명령어 부분이다. 이후, 같은 함수에서 setup\_stack 이후 부분에서 실제로 esp를 옮겨가며 스택에 쌓는다. 각 argument와, 그 주소, 개수, 리턴 주소를 쌓고, 필요한 경우라면 word-align을 해준다.

다음으로, userprog/syscall.c에서 system call을 구현하였다. 이 과정에서 check\_address라는 함수를 작성하였다. 이는 포인터가 NULL인지, 유저 영역인지, 커널 영역인지, unmapped된 메모리인지 확인하는 역할이다. syscall\_handler에서는 SYSCALL이 어떤 것인지 확인 한 후, 실행한다. 만약 리턴값이 있는 함수라면 eax에 리턴 주소를 저장하여 준다. 또, 같은 파일에서 6개의 system call 함수들과, 2개의 추가 함수들을 구현하여 주었다. 이 과정에서 부모와 자식 프로세스 간의 관계를 알 수 있는 구조체 변수들을 thread에 추가하고, init하는 함수에 해당 변수들을 초기화 해주는 코드들도 추가하였다. 이 내용은 threads 내의 thread.c, thread.h에 구현하였다. 추가적인 함수 구현 중, max\_of\_four\_int 함수에서는 인자가 4개이므로 이를 위한 syscall4를 lib/user/syscall.c에 작성하였고, 추가 함수들이 실행될 수 있도록 makefile 수정, additional.c 함수 작성 등의 작업을 하였다.

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

도표, 텍스트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

스케치, 그림, 도표, 종이접기이(가) 표시된 사진

자동 생성된 설명

1. System Calls

도표, 평면도, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명

이를 전체 그림으로 나타내면 다음과 같다.

도표, 텍스트, 기술 도면, 스케치이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**

1. Argument Passing

먼저 userprog/process.c의 load함수에서 file\_name을 parsing하는 부분이다.

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

자동 생성된 설명

공백을 기준으로 parsing을 해주었으며, 처음에는 2개 이상의 공백을 다루지 않았었는데, test case에서 그러한 입력이 들어와 fail하는 것을 알아내어 추가하였다. 이는 이전 칸도 공백이 아닐 때만 arg개수를 늘리는 식으로 해결하였다. 다음으로는 스택에 쌓는 부분이다.   
텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위에서 파악한 arg 개수만큼 반복문을 순회하며 해당 단어들을 argv[i]에 저장한다(405~406번 줄). 이때, 392번째 줄의 if문은 위에서 말한 연속된 공백을 걸러준다. 또, add라는 배열에 단어들을 저장한 주소를 저장하여 준다.

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

자동 생성된 설명

그 후, word-align이 필요하다면 이를 맞춰주고, add에 저장했던 argv 주소들도 쌓아준다. 마지막으로 arg개수와 return address를 쌓아준다. 그 후, 동적 할당했던 메모리들을 free해주며 스택에 쌓는 과정이 끝이난다.

1. User Memory Access

threads/vaddr.h에 있는 함수들을 이용하여 비정상적인 메모리 접근을 하는 것을 판별하고, 비정상적인 메모리 접근이라면 exit(-1)을 해주는 함수를 syscall.c에 작성하였다. 아래 함수에서 cnt를 받는 이유는, 인자 개수에 따라 검사하는 개수가 달라지기 때문이다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

또, 테스트 중 페이지 폴트 오류가 계속 발생하였다. 이는 userprog/exception.c의 page\_fault함수에 아래와 같은 코드를 추가해 비정상 종료를 시켜주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

존재하지 않는 페이지거나, 유저가 아니거나, 유저 주소가 아니거나, 커널 주소라면 비정상 종료를 시켜주는 코드이다.

1. System Calls

가장 먼저 userprog/syscall.c 파일을 살펴보자. syscall\_handler에서 check\_address함수를 통해 주소가 유효한지 검사하고, 해당하는 함수를 호출하였다. 또, 이 과정에서 return 값이 있다면 eax에 저장하여 주었다.

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

자동 생성된 설명

다음은, 6개의 함수 구현 부분이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

halt에서는 매뉴얼의 설명대로 shutdown\_power\_off()함수를 호출하여 처리하였고, exit에서는 명세서에 나와있는 출력메시지를 출력하고 thread status를 변경해준 뒤, thread\_exit()을 호출하였다. exec에서는 process\_excute함수를 호출하며 리턴해 주었다. wait에서는 process\_wait함수를 호출하며 리턴해 주었다. process\_wait 함수에 대해서는 잠시 뒤 더 살펴볼 것이다. read에서는 stdin에 대해서만 buffer에 입력을 한 뒤, 개수를 리턴하여 주었고, write에서는 stdout에 대해서만 putbuf를 이용해 write해주고 size를 리턴하여 주었다.

wait에서 호출한 process\_wait함수를 살펴보자. 이는 process.c에 있다. 기존에는 단순 반복문을 통해서 process\_wait을 임시 방편으로 사용하였다. 이를 semaphore를 이용하여 다시 구현하였다. process\_wait함수에선 자식이 죽을 때까지 기다리고, 죽게 되면 thread exit\_status를 리턴할 것이다. 자식을 찾으면, sema\_down하여 기다리고, load\_sema를 sema\_up하여 메모리를 삭제 전까지 남겨둔다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이제 exit에서는 위 과정을 반대로 진행해주면 된다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이제 여기서 사용한 변수들을 추가한 threads/thread.h를 살펴보자.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위와 같은 변수들을 추가하였다. 추가한 변수들도 init\_thread에서 초기화가 잘 이루어져야 하므로, threads/thread.c에서 아래와 같은 코드를 추가해 주었다.

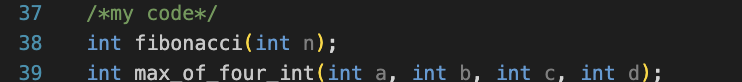
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Additional System calls

명세서에 나와있는 대로 lib/user/syscall.h, lib/user/syscall.c, lib/syscall-nr.h, userprog/syscall.h, userprog/syscall.c, makefile을 수정하여 주었다. 또, examples에 additional.c 파일을 만들어 구현하여 주었다.

lib/user/syscall.h에 두 함수의 프로토타입을 선언하였다.



lib/user/syscall.c에 max\_of\_four\_int 함수를 위한 syscall4를 추가 구현하였다. 이는 syscall2, 3을 참고하였다. 또, 두 함수를 추가하여 각각 인자 개수에 맞는 syscall1, syscall4로 인자를 넘겨준다.

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

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

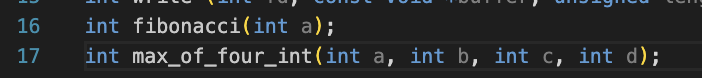
자동 생성된 설명

lib/syscall-nr.h에서는 enum에 두 함수 콜 넘버를 추가해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

userprog/syscall.h에도 두 함수의 프로토 타입을 추가해 주었다.



userprog/syscall.c에서는 syscall\_handler에서 추가한 콜 넘버에 따른 함수 실행하는 부분을 추가하고, 실제로 두 함수의 내용을 구현해 준다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

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

자동 생성된 설명

테스트를 하기 위해 examples/additional.c를 아래와 같이 작성하여 각 추가함수의 출력을 테스트할 수 있게 해주었다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

atoi를 위해 stdlib.h도 추가하였다.

examples/Makefile에도 additional.c가 실행될 수 있게 아래 코드에서 7번 줄에 additional을 추가하고, 20번째 줄을 추가하였다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

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

10 20 62 40에 대한 결과인 55, 62가 잘 출력되는 모습이다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

7, 7, 8, 2에 대한 출력도 아래와 같이 잘 나오는 것을 확인할 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

마지막으로, 15 20 40 54에 대해서도 15번째 피보나치 수인 610과 네 수 중 최댓값인 54가 잘 출력되는 것을 확인할 수 있다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

또한, 프로젝트 1에서 요구하는 것 모두 충족하는 것을 확인할 수 있다.

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

자동 생성된 설명