시스템 콜 추가 및 이해

(Practical understanding of System Call through Implementation)

컴퓨터학과 2013210063 문기준

제출일:2019.04.14. Freeday Used: 0

Korea University

Contents

개발환경 / 시스템 콜이란? / 소스 코드 / 실행 결과 / Problem & Solution

개발환경

해당 프로젝트의 개발환경은 VM Virtual box위에 설치한 리눅스(Ubuntu 18.04.2 LTS(64bit))을 기반으로 진행되었다. 해당 개발환경을 설정한 이유에 대해 3가지 질문을 던질 수 있다**. 1. 왜 VM Virtual box위에서 진행되어야 하는가? 2. 왜 윈도우가 아닌 리눅스인가? 3. 왜 굳이 LTS가 붙은 해당 버전의 Ubuntu를 사용하는가?** 이다. 이 질문에 대해 각각의 이유를 설명하겠다.

VM Virtual box에서 VM(Virtual Machine)이란 현재 이용하고 있는 하드웨어와 직접적인 연관이 없는 가상 컴퓨터를 의미한다. Virtual box는 물리 컴퓨터 위에 직접 설치되는 Host OS위에 Hypervisor의 역할을 수행하여 가상 머신 위에 독립된Guest OS를 구축할 수 있게 도와준다. 이에 물리 컴퓨터에 물리적인 시스템을 조작하는 OS의 kernel을 프로그래밍 하는 본 프로젝트에 성향상 사용하고 있는 컴퓨터에 변조를 방지하기 위함이라 할 수 있다.

그렇다면 왜 원도우가 아닌 리눅스인가? 윈도우는 OS에 대한 소스가 리눅스에 비해 상당히 제한적이다. 리눅스는 완벽한 오픈 소스 프로젝트로 업데이트 및 개발에 대한 배포 및 수정이 자유롭다. 즉, OS의 그래픽 인터페이스부터 Kernel까지의 거의 모든 소스 코드에 접근 및 수정이 가능하다는 것이다. 이에 OS단에서의 Kernel을 프로그래밍 하려는 본 프로젝트에 리눅스가 보다 적합하다. 또한 OS에 대한 라이선스를 돈으로 지불해야 하는 원도우에 비해 리눅스는 완벽히 무료이기 때문에 더욱 그 선호도가 올라간다.

우선 ubuntu는 데스크탑과 서버에서 가장 많이 쓰이는 리눅스 배포판이기에 ubuntu를 사용을 한다. 그 중에서도 LTS(Long-term support)가 붙은 ubuntu는 수 많은 우분투 버전들 중에 안정성과 보안성을 지원해주는 장기 지원 버전으로 평균 5년 정도의 표준 지원을 받을 수 있다. Ubuntu 18.04.2버전의 경우 2019년 2월 15일에 릴리즈된 최신 버전으로 LTS가 붙은 ubuntu중에 최신에 속하는 버전이기에 해당 버전을 사용하였다.

리눅스의 시스템 콜

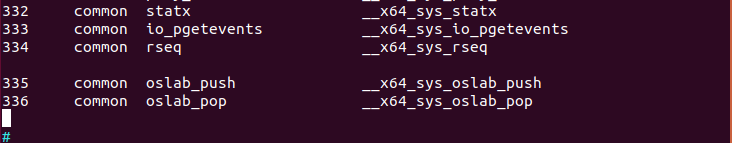
1. 시스템 콜 이란?

운영체제는 사용자 프로그램이 시스템 자원을 직접적으로 사용하도록 허용하지 않는다. 즉, 시스템 자원을 사용하는 여러가지 기능들은 운영체제가 제공해주고 일반사용자나 사용자 프로그램들은 그것을 요청해야 하는데 그 요청하는 것이 바로 시스템 콜이다. 더 정확히 이야기하자면 시스템 콜이란 운영체제가 user mode에서 kernel mode로 전환해 주는 프로그래밍 인터페이스라는 것이다. 이는 컴퓨터 자원의 하드웨어 동작을 직접 구체적으로 알지 못하더라도 그것을 편리하게 이용할 수 있게 되는 것이다. 이렇듯 사용자 프로그램인 user mode에서 시스템 자원을 관리하는 kernel mode로 직접 접근이 허용되지 않는 이유는 두가지 설계적인 관점이 있다. 첫번째, kernel은 컴퓨터 자체의 시스템 자원을 관리하는 곳이기에 user mode와 항상 독립적이어야 하며 user mode의 break down이 kernel에 영향을 미치지 못하게 하는 안정성을 유지하기 위해서 이다. 두번째, user mode에서 kernel mode에 직접 접근이 쉬워진다면 컴퓨터 자체의 보안성 문제가 생길 수 있기 때문이다.

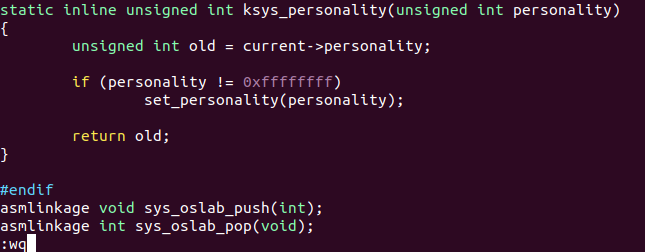
1. 시스템 콜의 호출 원리

시스템 콜의 호출 루틴은 항상 User mode와 Kernel mode를 왕복하는 루틴을 가지고 있다. 먼저 유저 프로그램에 syscall()이라는 매크로 함수가 시스템 콜 번호와 인자를 받는다. 이렇게 호출된 syscall함수는 <unistd.h>헤더(운영체제(POSIX) API에 접근을 하게 해주는 헤더)는 syscall table에서의 시스템 콜의 고유 번호를 찾아System Call interface를 통해System Call interrupt가 발생하여 user mode에서 kernel mode 로 진입하게 된다. 여기서 사용자 프로세스는 일시 정지한 상태이다. 이 과정에 있어서 <unistd.h>헤더는 과제에서 수정한 syscall\_64.tbl과 syscall.h와 Makefile이 포함된kernel을 build하는 과정을 통해 만들어진 syscall table을 기반으로 한다. 다음은 Core kernel이 syscall table에서 해당 call을 찾는다. 이후 Kernel 모듈이 해당 시스템 콜의 소스코드를 통해syscall action을 수행하여 수행한 결과 값을 반환한다. 이는 Core Kernel이 user프로세스에 리턴해 주면서 user program이 다시 실행되게 되는 것이다.

작성한 모든 소스코드

1. **syscall\_64.tbl**

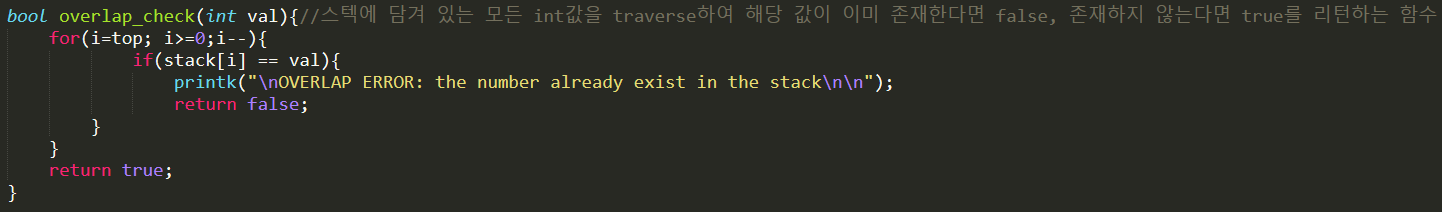
userspace application을 위한 시스템 콜 고유번호를 설정한다.

1. **syscalls.h**

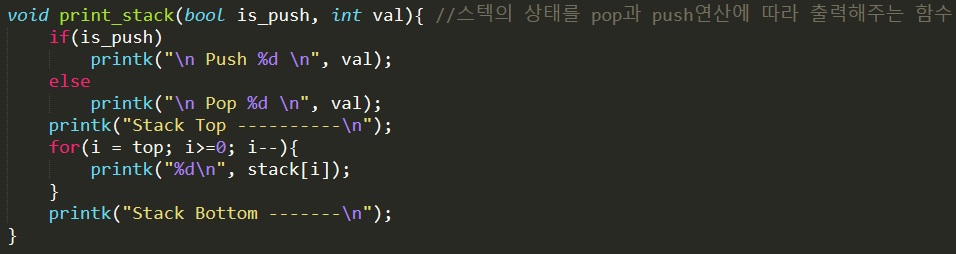
asmlinkage를 이용하여 어셈블리 코드에서 직접 시스템 콜을 호출할 수 있게 시스템 콜의 프로토타입을 정의한다.

1. **my\_stack\_syscall.c(설명의 간결성을 위해 소스코드의 중점적인 요소만 설명)**

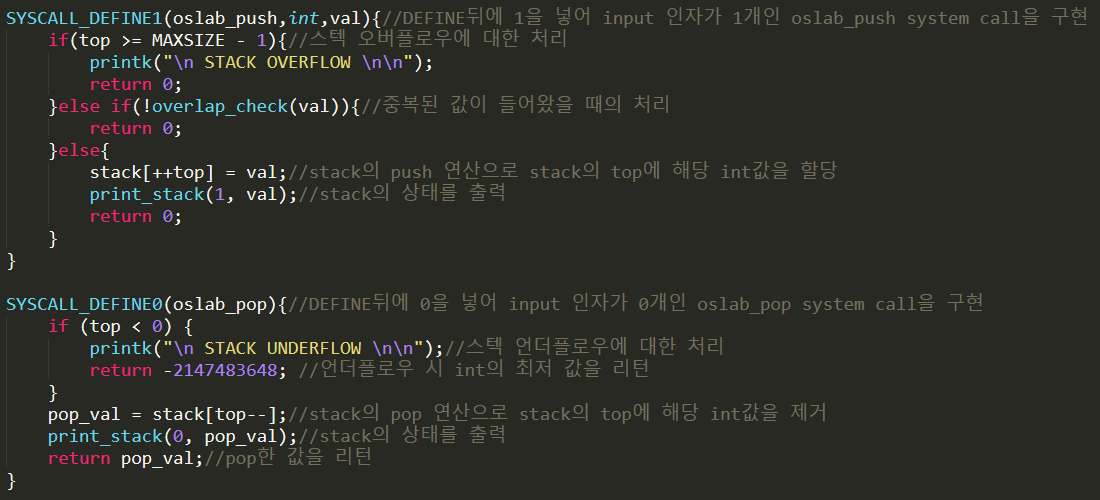
정의한 시스템 콜 자체의 기능을 직접 구현하는 파일로 과제에서 제시하는 stack의 pop과 push를 구현하는 소스 코드이다.



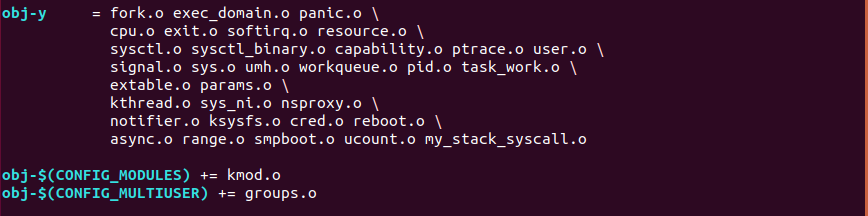
과제의 중점적인 내용이었던 중복 체크에 대한 내용을 보다 직관적으로 처리하기 위해 함수를 따로 정의하여 중복으로 들어가는 수에 대한 처리를 구현하였다.



또한 중복적으로 pop과 push를 통해 스텍의 현 상태를 출력해야 하는 부분이 겹침으로 이를 별도의 함수로 구현하여 가독성을 높혔다.



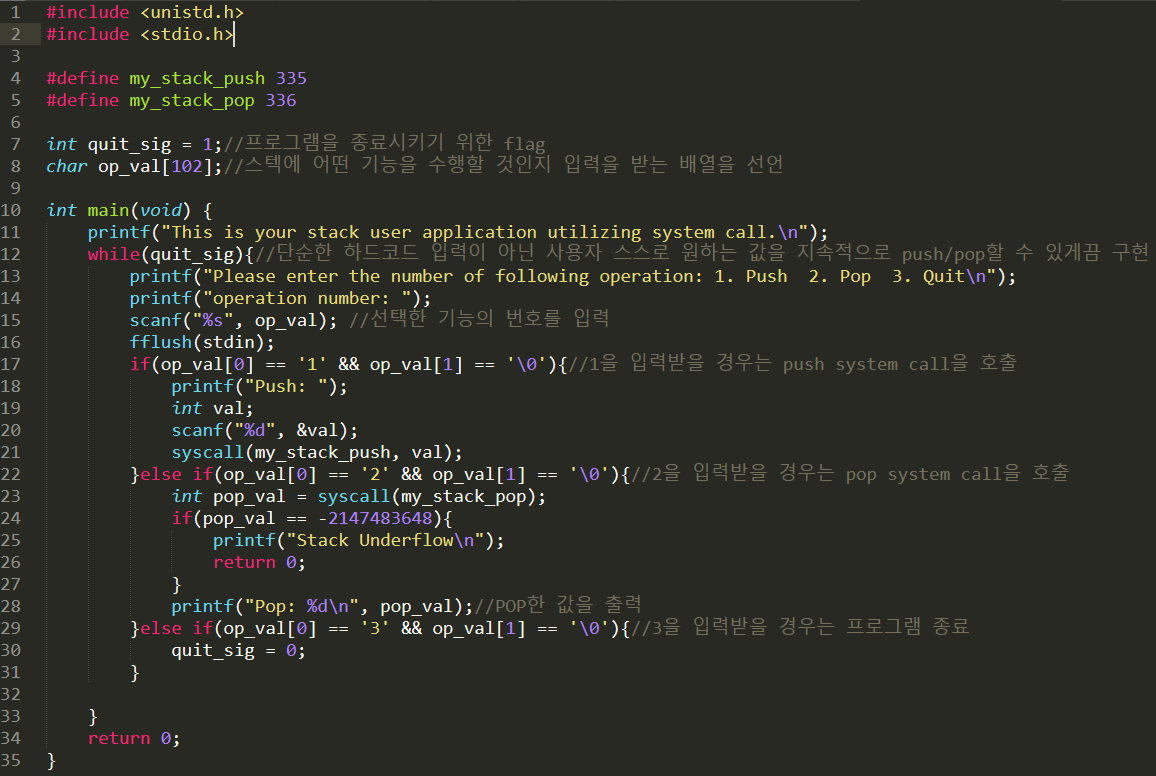
전형적인 C에서의 스텍 구현을 배열을 통해 구현하였으며 system call을 구현하기 위한 매크로인 SYSCALL\_DEFINEx를 이용하여 각 시스템 콜이 받는 파라미터의 개수를 지정하였고 스텍이 가지고 있는 결정적인 오류인 오버플로우와 언더프로우에 대한 구현 또한 더 하였다. 이어 과제에서 제시하는 중복 처리에 대한 내용을 별도로 정의한 함수를 이용하여 보다 간결한 코드를 구현하였다.

1. Makefile

이것은 Kernel에 해당 시스템 콜을 추가하는 마지막 작업으로 시스템 콜의 실제 기능이 들어있는 C파일을 Kernel의 make 과정에 포함시키기 위해 obj-y 부분에 추가해 주는 작업이다. 여기에 .c가 아닌 .o가 들어가는 것은 make 과정에 있어 소스코드를 컴파일하여 만든 실행 파일을 이용하여 시스템 콜을 execution하기 때문이다.

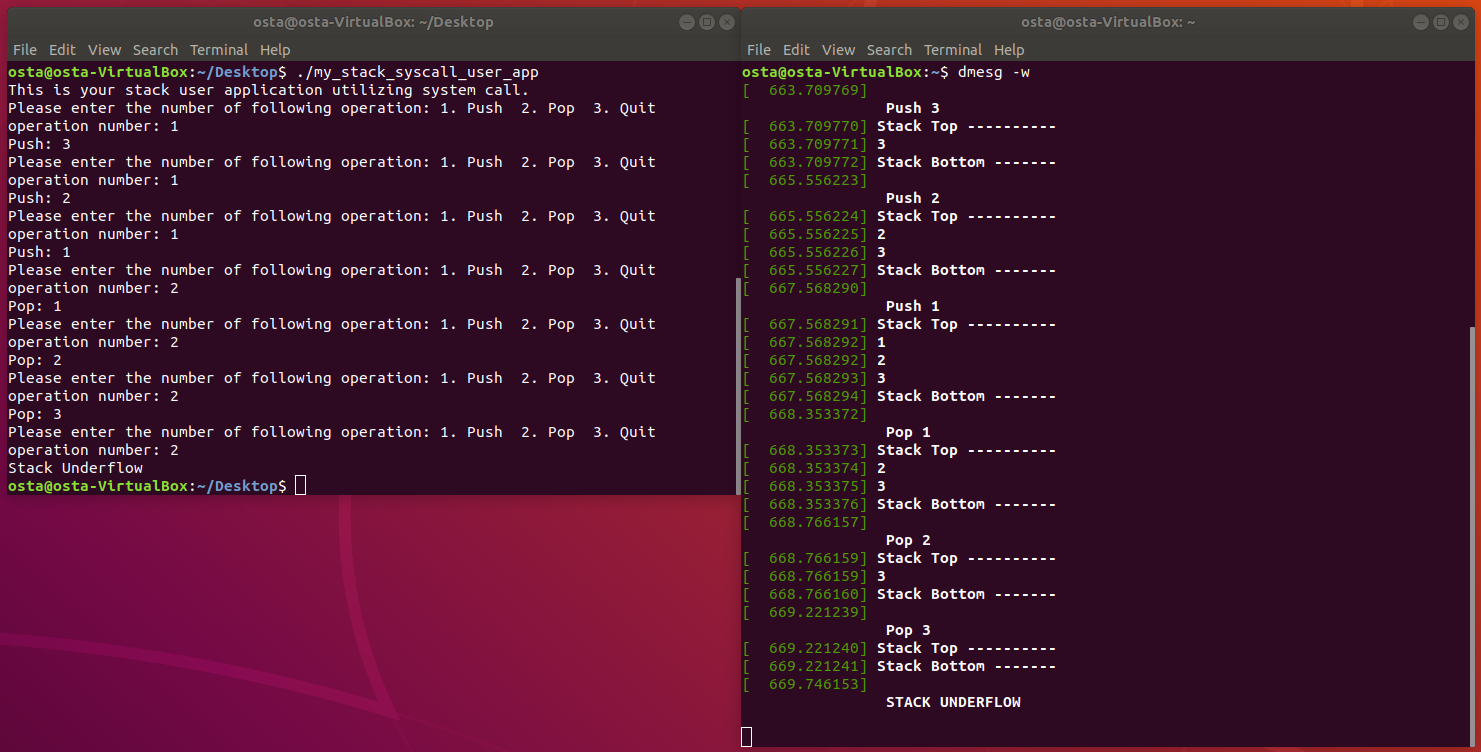
1. 유저 Application (my\_stack\_syscall\_user\_app.c)

유저 Application은 user-space에서 프로그램을 통해 system call을 호출하는 프로그램으로 syscall() 이라는 매크로 함수를 이용하여 <unistd.h> 헤더를 통해 시스템 콜을 system call interrupt로 kernel에서 불러온다.



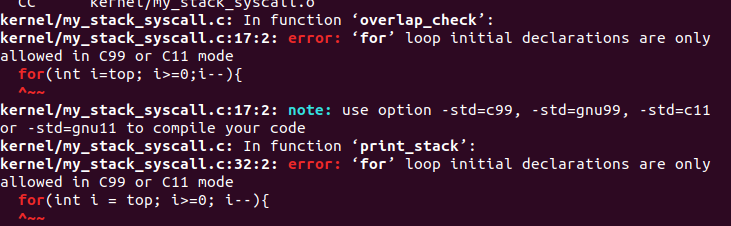
현재 과제에서 답안으로 제시하고 있는 수동적인 입력에 대한 실행에 대비하여 보다 능동적인 프로그램을 만들어 보았다. 사용자는 해당 프로그램을 통해 스텍에 자신이 원하는 데로push/pop 시스템 콜을 호출할 수 있다. 또한 시스템 콜의 정의에 따라 반환 값이 있는 pop의 경우 int pop\_val변수를 통해서 시스템 콜에서 pop한 변수를 반환하여 출력한다.

실행 결과 스냅샷

1. Youtube 영상: <https://www.youtube.com/watch?v=wBkze06cLbs>
2. 스냅샷:

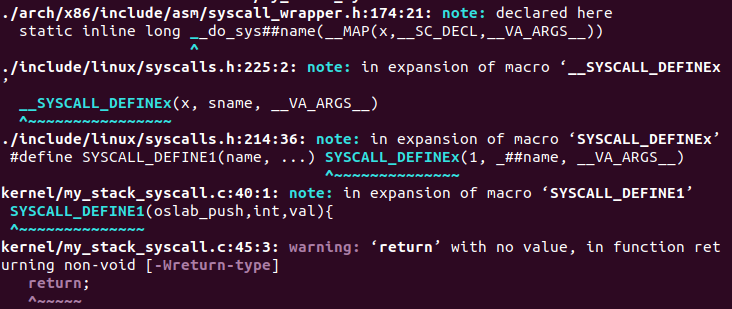
숙제 수행 과정 중 발생한 문제점과 해결방법

생각보다 간단해 보였던 이번 과제가 어렵게 느껴졌던 것은 아래와 같은 문제점들 때문이었다. 같은 C언어로 구현을 하지만Kernel상에서의 프로그램인 만큼 user-space에서의 프로그램과 많이 다르다는 것을 알았다. 이는 user-space에서의 C standard library의 편의성을 깨닫는 좋은 계기였다.

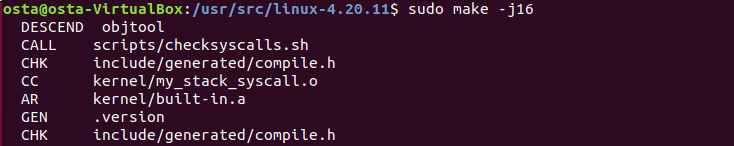
문제점 1 : my\_stack\_syscall.c 의 시스템 콜 함수에 대한 for문 안에서의 변수 초기화 에러

이렇듯 user-space 프로그램에서는 당연듯이 써왔던 for문 상에서의 변수 초기화에 대한 에러가 나는 것을 볼 수 있다.

해결방법: 이를 해결하기 위하여 int i를 전역 변수로 선언해주었다.

문제점 2 : my\_stack\_syscall.c 의 시스템 콜 함수에 대한 return 값 에러

확실한 컴파일 에러는 아니었지만 SYSCALL\_DEFINEx함수 안에 return 값을 적절히 부여해 주지 않아 warning debug log가 남는 것을 볼 수 있었다.

해결방법: 이는 return 0;를 선언해주는 것을 통해 warning debug log를 없애 주었다.

이렇듯 아무런 에러와 warning이 없이 make가 되는 것을 확인하였다.

문제점 3 : my\_stack\_syscall.c 의 수정 사항이 반영되는 기준에 대한 문제점

아마 과제를 수행하는 중 제일 과제 수행 시간을 잡아먹었던 것이 바로 이 문제라고 할 수 있겠다. 시스템 콜의 코드의 에러를 수정하고 make install을 해도 dmesg상의 log가 전혀 달라지지 않는 다는 것을 알게 되었다.

해결방법: 이 이유에 대해 깊게 생각하다 보니 0차 과제에서 kernel을 install하고 reboot하는 과정이

있다는 것을 알게 되었고 reboot을 통해서 수정 사항을 제대로 반영하여 code를 수정해 나갔다.

결론: 이번 과제를 통해서 무엇보다 이론을 통해서 배웠던 시스템 콜을 직접 구현해 봄으로서 추상적이었던 시스템콜을 구체적으로 알게 되었으며 무엇보다 앞으로 있을 kernel 상의 프로그래밍에 있어 많은 요령을 얻어가는 좋은 기회였다.