


운영체제 과제 1

컴퓨터학과 2019320110 정우성
제출: 2023 04 09 (freeday 0 일)

0. 개발 환경

Window 11 Pro + Virtual box 7.0.6 + Ubuntu 18.04.2

 장치 사양

장치 이름	DESKTOP-0OOCFA8
프로세서	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
설치된 RAM	16.0GB(15.8GB 사용 가능)
장치 ID	929EBDF4-0FC5-4617-B2D8-685338A34093
제품 ID	00330-51657-48593-AAOEM
시스템 종류	64비트 운영 체제, x64 기반 프로세서
펜 및 터치	이 디스플레이에 사용할 수 있는 펜 또는 터치식 입력이 없습니다.

관련 링크 도메인 또는 작업 영역 시스템 보호 고급 시스템 설정

 Windows 사양

에디션	Windows 11 Pro
버전	22H2
설치 날짜	2023-04-08
OS 빌드	22621.1413
경험	Windows Feature Experience Pack 1000.22639.1000.0

Figure 1 : 개발 환경

1. 과제 설명

a. 시스템 콜?

System call 이란, 수업시간에도 배웠듯이 user mode 에서 kernel mode 로 진입하는 api 다. Kernel 에 구현된 기능들이 필요한 경우 system call 을 이용해 모드를 바꾸고, 구현된 system call handler 을 활용해 원하는 작업을 수행한다.

우리 과제에서는 user application 에서 syscall 매크로 함수를 사용해 우리가 원하는 system call handler 를 작동시킨다.

b. syscall()

unistd.h 에 구현된 system call 을 호출할 수 있는 Wrapper 함수다. 호출하고 싶은 system call 의 번호를 첫번째 인자로 넣고, 두 번째 인자부터 system call 의 handler 가 사용할 인자들을 넣는다.

system call 을 호출하기 위해 syscall()은 어셈블리어 int 0x80 명령어를 사용한다. (아닌 경우도 있긴 한 듯)

c. INT 0x80?

Interrupt 0x80 번을 실행하겠다는 의미. Interrupt 들은 interrupt vector table 에 정의되어 있다. Linux 에서 이 번호는 system call 을 호출하겠다는 인터럽트로 지정되어있다. 따라서 0x80 번에 지정된 인터럽트 핸들러가 동작하게 되는데, 이 핸들러는 전달받은 번호에 해당하는 system call 을 IVT 에서 찾아 실행하는 역할을 한다. 이 IVT 에 해당하는 것이 과제에서 건드린 syscall_64.tbl 이라고 이해하였다.

```
344 333 common  to_pgetevents      __x64_sys_to_pgetevents
345 334 common  rseq              __x64_sys_rseq
346
347 #oslab
348 335 common  os2023_push        __x64_sys_os2023_push
349 336 common  os2023_pop         __x64_sys_os2023_pop
350
351 #
352 # x32-specific system call numbers start at 512 to avoid
```

위와 같이 해당 파일을 수정했다.

d. asmlinkage?

우리가 만드는 system call 함수들을 선언할 때 prefix 로 asmlinkage 라는 modifier 를 붙였다. 이는 우리가 syscall()로 system call handler 에게 전달하는 함수 인자를 레지스터를 이용해 전달하지 말고 메모리 스택을 사용하라고 명시하는 의미이다.

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error	Notes
alpha	callsys	v0	v0	a4	a3	1, 6
arc	trap0	r8	r0	-	-	
arm/OABI	swi NR	-	r0	-	-	2
arm/EABI	swi 0x0	r7	r0	r1	-	
arm64	svc #0	w8	x0	x1	-	
blackfin	excpt 0x0	P0	R0	-	-	
i386	int \$0x80	eax	eax	edx	-	
ia64	break 0x100000	r15	r8	r9	r10	1, 6
m68k	trap #0	d0	d0	-	-	
microblaze	brki r14,8	r12	r3	-	-	
mips	syscall	v0	v0	v1	a3	1, 6
nios2	trap	r2	r2	-	r7	
parisc	ble 0x100(%sr2, %r0)	r20	r28	-	-	
powerpc	sc	r0	r3	-	r0	1
powerpc64	sc	r0	r3	-	cr0.S0	1
riscv	ecall	a7	a0	a1	-	
s390	svc 0	r1	r2	r3	-	3
s390x	svc 0	r1	r2	r3	-	3

Figure 3: architecture calling conventions

아키텍처마다 위 그림처럼 커널에게 인자를 보내고 결과를 받는 레지스터가 정해져 있다. 이를 사용하지 말고, 메모리를 사용하라고 명시하려는 의도이다.

왜일까? syscall()이 실행시킨 INT 0x80 핸들러는 어셈블리 코드다. 여기서 syscall table 에 해당하는 컴파일된 C 코드를 찾아서 직접 실행을 하게 된다. gcc 는 레지스터를 사용하든 스택을 사용하든 보다 빠른 방법으로 함수의 인자를 저장 / 사용하게끔 컴파일할 텐데, 어셈블리단에서 사용하는 인자 전달 방식과 다를 수 있다. 이 때 두 기계어 간 불일치를 방지하기 위해 asmlinkage 를 사용하여 무조건 스택을 이용해 인자를 주고 받게끔 하는 것이다.

e. System call 구현

그렇게 만들어진 테이블 / 프로토타입을 바탕으로 kernel 에 함수를 아래와 같이 구현했다.

간단한 스택이기 때문에 STACK_MAX 칸의 int 배열을 사용했다. push / pop 을 효과적으로 하기 위해 stack top 의 index 를 top_idx 라는 전역 변수에 저장하여 사용했다. push 는 STACK_MAX 개까지만 가능하고, pop 은 스택이 비어있는 경우 실행되지 않는다. 과제 설명에 쓰여진 것처럼 duplicate 데이터는 push 되지 않는다.

각 작업이 완료된 후 스택의 로그를 printk 함수를 사용해 남긴다.
커널의 makefile 에 해당 커널 코드를 추가해주었다.

f. SYSCALL_DEFINEx

구현을 위해 SYSCALL_DEFINEx 매크로를 사용했다. x 는 해당 system call 이 사용하는 인자의 개수를 의미한다.

```
234 #ifndef SYSCALL_DEFINEx
235 #define SYSCALL_DEFINEx(x, name, ...) \
236     __diag_push(); \
237     __diag_ignore(GCC, 8, "-Wattribute-alias", \
238         "Type aliasing is used to sanitize syscall arguments"); \
239     asm_linkage long sys##name(__MAP(x, __SC_DECL, __VA_ARGS__)) \
240     __attribute__((alias(__stringify(__se_sys##name)))); \
241     ALLOW_ERROR_INJECTION(sys##name, ERRNO); \
242     static inline long __do_sys##name(__MAP(x, __SC_DECL, __VA_ARGS__)); \
243     asm_linkage long __se_sys##name(__MAP(x, __SC_LONG, __VA_ARGS__)); \
244     asm_linkage long __se_sys##name(__MAP(x, __SC_LONG, __VA_ARGS__)) \
245     { \
246         long ret = __do_sys##name(__MAP(x, __SC_CAST, __VA_ARGS__)); \
247         __MAP(x, __SC_TEST, __VA_ARGS__); \
248         __PROTECT(x, ret, __MAP(x, __SC_ARGS, __VA_ARGS__)); \
249         return ret; \
250     } \
251     __diag_pop(); \
252     static inline long __do_sys##name(__MAP(x, __SC_DECL, __VA_ARGS__)) \
253 #endif /* __SYSCALL_DEFINEx */
254
```

syscalls.h 내의 구현을 따라가다보면 위 코드를 발견할 수 있다. 이는 첫 인자로 주어진 ##name 을 이용해 sys_##name 을 이름으로 하는 system call 을 만든다는 의미다. 함수의 인자로 타입 / 데이터 변수 를 순서에 맞게 ##name 뒤에 적는다.

따라서 sys_os2023_push(int a) 와 sys_os2023_pop()이 만들어졌다.

g. User application

(a) 에 명시한 대로 syscall() 매크로 함수를 이용해 직접 만든 system call 을 호출하였다.

```
osta@osta-VirtualBox:~/Desktop/KU/COSE341_OS$ ./oslab_call_stack
Push 1
Push 1
Push 2
Push 3
Pop 3
Pop 2
Pop 1
```

결과는 위와 같다.

```
[ 35.497531] [System Call] os2023_push :
[ 35.497532] Stack Top -----
[ 35.497534] 1
[ 35.497534] Stack Bottom -----
[ 35.497625] [System Call] os2023_push :
[ 35.497626] Stack Top -----
[ 35.497626] 1
[ 35.497627] Stack Bottom -----
[ 35.497629] [System Call] os2023_push :
[ 35.497629] Stack Top -----
[ 35.497629] 2
[ 35.497630] 1
[ 35.497630] Stack Bottom -----
[ 35.497632] [System Call] os2023_push :
[ 35.497632] Stack Top -----
[ 35.497632] 3
[ 35.497633] 2
[ 35.497633] 1
[ 35.497633] Stack Bottom -----
[ 35.497635] [System Call] os2023_pop :
[ 35.497635] Stack Top -----
[ 35.497636] 2
[ 35.497636] 1
[ 35.497636] Stack Bottom -----
[ 35.497638] [System Call] os2023_pop :
[ 35.497638] Stack Top -----
[ 35.497639] 1
[ 35.497639] Stack Bottom -----
[ 35.497640] [System Call] os2023_pop :
[ 35.497641] Stack Top -----
[ 35.497641] Stack Bottom -----
osta@osta-VirtualBox:~/Desktop/KU/COSE341_OS$
```

dmesg 를 이용해 확인한 printk 로그는 위와 같다.

2. 문제들

1. 노트북을 재부팅 하면 Virtual box manager 가 VM 을 켜지 못했다
hypervisor 도 다 제대로 되어 있었고 별 짓 다 해봤으나 virtual box manager 를 C 드라이브에 재설치했더니 됐다. 신기했던 점은 manager 를 삭제해도 VM 은 삭제되지 않았다는 것? 다시 커널 컴파일을 하지 않아도 되어 시간을 줄였다.
2. Syscall_64.tbi 이 readonly 라서 수정이 안 된다
 - a. Chmod 777 로 해결했다.
 - b. 근데 ls -l 을 이용해 확인해보니 관리자에겐 w 권한이 있었다. 이후 모든 파일을 수정할 때 sudo vim 을 사용해 chmod 를 일일이 사용하지 않았다.

3. 참고한 자료들

- <https://man7.org/linux/man-pages/man2/syscall.2.html>
- <https://stackoverflow.com/questions/1817577/what-does-int-0x80-mean-in-assembly-code>
- <https://stackoverflow.com/questions/10060168/is-asmlinkage-required-for-a-c-function-to-be-called-from-assembly>
- <https://stackoverflow.com/a/10063603>
- <http://egloos.zum.com/rousalome/v/9991442>
-