Strace를 사용한 리눅스 ls 명령어 분석

2025-07-15

최용진 작성

목차

1. ls 명령어 개요

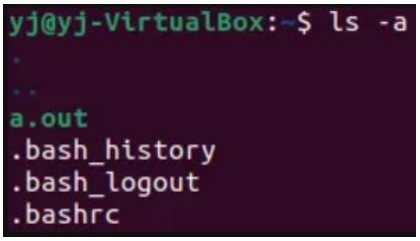
ls는 리눅스에서 디렉토리의 내용을 나열하는 기본적인 명령어로, **파일 및 디렉토리 목록을 출력하는 데 사용된다.**

`$ ls [옵션] [경로]`

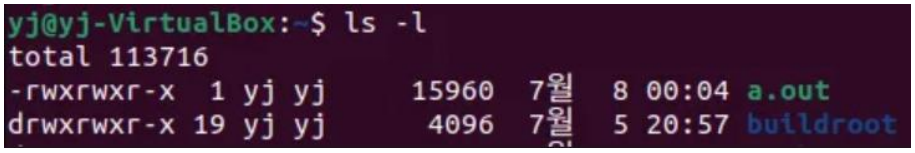
옵션과 경로는 생략 가능하며, 생략시 현재 디렉토리의 내용만 보여준다.

주요 옵션:

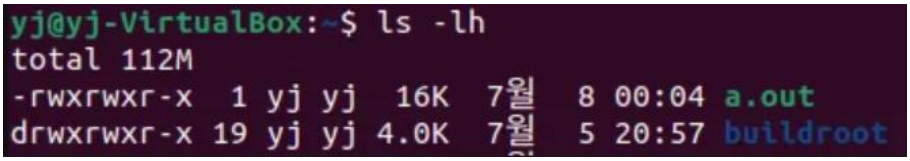
- -a: 숨김파일까지 모두 표시



- -l: 권한, 소유자, 크기 등의 상세 정보 열 형식 출력



- -h: 읽기 쉬운 파일 크기로 표시 (보통 -l과 함께 사용)



- -S: 파일 크기 기준 정렬

2. strace 개요

strace는 리눅스에서 프로세스가 호출하는 시스템 콜을 추적하는 명령어이다.

프로그램이 커널에 어떤 요청을 하는지를 실시간으로 보여준다.

`$ strace [명령어]`

주요 옵션:

- `-e trace=read,write`: 특정 시스템 콜만 추적
- `-T`: 각 시스템 콜의 실행 시간 표시
- `-o [파일명]`: 결과를 파일로 저장
- `-p [PID]`: 이미 실행 중인 프로세스 추적

3. 시스템 콜 추적의 필요성

시스템 콜은 유저 공간과 커널 공간 간의 인터페이스이며, 리눅스 명령어가 커널에 요청을 보낼 때 필수적으로 사용된다. 어떤 시스템 콜이 호출되는지 알면 다음과 같은 점에서 유용하다:

1. 명령어의 내부 동작 원리를 이해할 수 있다.
2. 성능 병목 지점이나 오류 원인을 추적할 수 있다.
3. 동일한 기능을 시스템 프로그래밍으로 구현할 때 참고할 수 있다.

4. strace를 통한 ls 명령어 분석

strace ls 명령을 통해 ls 명령어 실행 시 호출되는 시스템 콜을 추적하였다. 결과는 다음과 같다.

```
yj@yj-VirtualBox:~$ strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffd5e29b950 /* 53 vars */) = 0
brk(NULL)                               = 0x5dc475561000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe2c5387d0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7041fefcc000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=55504, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 55504, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7041fefbe000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=166280, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 177672, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7041fef92000
mprotect(0x7041fef98000, 139264, PROT_NONE) = 0
mmap(0x7041fef98000, 106496, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7041fef98000
mmap(0x7041fefb2000, 28672, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20000) = 0x7041fefb2000
mmap(0x7041fefba000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x27000) = 0x7041fefba000
mmap(0x7041fefbc000, 5640, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7041fefbc000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\325\31p\226\367\t\200\30)\261\30\257\33|\366c"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7041fec00000
mprotect(0x7041fec28000, 2023424, PROT_NONE) = 0
mmap(0x7041fec28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7041fec28000
mmap(0x7041fedbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7041fedbd000
mmap(0x7041fee16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7041fee16000
mmap(0x7041fee1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7041fee1c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre2-8.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=613064, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 615184, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7041feefb000
mmap(0x7041feefd000, 438272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7041feefd000
mmap(0x7041fef68000, 163840, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6d000) = 0x7041fef68000
mmap(0x7041fef90000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x94000) = 0x7041fef90000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7041feef8000
arch_prctl(ARCH_SET_FS, 0x7041feef8800) = 0
set_tid_address(0x7041feef8ad0)         = 1726
set_robust_list(0x7041feef8ae0, 24)     = 0
rseq(0x7041feef91a0, 0x20, 0, 0x53053053) = 0
mprotect(0x7041fee16000, 16384, PROT_READ) = 0
mprotect(0x7041fef90000, 4096, PROT_READ) = 0
mprotect(0x7041fefba000, 4096, PROT_READ) = 0
mprotect(0x5dc446125000, 8192, PROT_READ) = 0
mprotect(0x7041ff006000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7041fefbe000, 55504)           = 0
statfs("/sys/fs/selinux", 0x7ffe2c538810) = -1 ENOENT (No such file or directory)
statfs("/selinux", 0x7ffe2c538810)      = -1 ENOENT (No such file or directory)
getrandom("\x72\xce\x10\x06\xe7\xf2\x53\x5c", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5dc475561000
brk(0x5dc475582000)                     = 0x5dc475582000
openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0444, st_size=0, ...}, AT_EMPTY_PATH) = 0
read(3, "nodev\tsysfs\nnodev\ttmpfs\nnodev\tbd"..., 1024) = 393
read(3, "", 1024)                       = 0
close(3)                                = 0
access("/etc/selinux/config", F_OK)     = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=7057008, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 7057008, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7041fe400000
close(3)                                = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
ioctl(1, TIOCGWINSZ, {ws_row=24, ws_col=80, ws_xpixel=0, ws_ypixel=0}) = 0
openat(AT_FDCWD, ".", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
newfstatat(3, "", {st_mode=S_IFDIR|0750, st_size=4096, ...}, AT_EMPTY_PATH) = 0
getdents64(3, 0x5dc475568ac0 /* 32 entries */, 32768) = 1112
getdents64(3, 0x5dc475568ac0 /* 0 entries */, 32768) = 0
close(3)                                = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
write(1, "a.out\t\t\t\t\t\t\t   linux\n", 21a.out                                linux
) = 21
write(1, "buildroot\t\t\t\t\t\t   Music\n", 24buildroot                           Music
) = 24
write(1, "Desktop\t\t\t\t\t\t   Pictures\n", 26Desktop                           Pictures
) = 26
write(1, "Documents\t\t\t\t\t\t   Public\n", 25Documents                          Public
) = 25
write(1, "Downloads\t\t\t\t\t\t   simple.c\n", 27Downloads                        simple.c
) = 27
write(1, "gcc-arm-10.3-2021.07-x86_64-aarc"..., 59gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu       snap
) = 59
write(1, "gcc-arm-10.3-2021.07-x86_64-aarc"..., 77gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu.tar.xz  strace_output.txt
) = 77
write(1, "hello\t\t\t\t\t\t\t   Templates\n", 25hello                            Templates
) = 25
write(1, "hello.c\t\t\t\t\t\t\t   Videos\n", 24hello.c                            Videos
) = 24
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

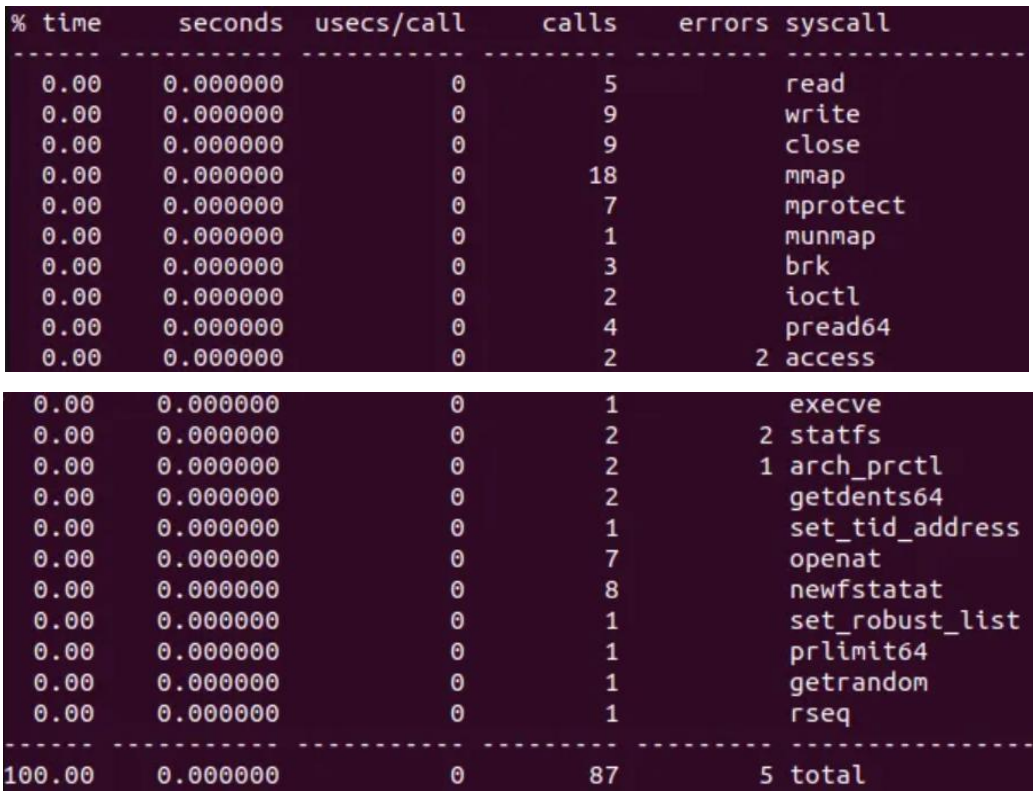
호출된 시스템 콜 정리

시스템 콜	역할	주요 인자	반환값 의미
read	파일 디스크립터로부터 데이터 읽기	fd, buf, count	읽은 바이트 수 or -1 (실패)
write	터미널(표준 출력)에 출력	fd(=1), buf, count	출력한 바이트 수
close	열린 파일 디스크립터 닫기	fd	0(성공), -1(실패)
mmap	파일/메모리 영역을 가상 메모리에 매핑	addr, length, prot, flags, fd, offset	매핑된 주소 or -1
mprotect	메모리 영역 접근 권한 설정	addr, len, prot	0 or -1
munmap	매핑된 메모리 해제	addr, length	0 or -1
brk	힙(프로그램 break 지점) 조정	addr	새 break 값 or -1
ioctl	터미널 등 디바이스 제어	fd, request, argp	0(성공), -1(실패)
pread64	특정 오프셋부터 읽기 (mmap 대체)	fd, buf, count, offset	읽은 바이트 수
access	파일 접근 가능 여부 검사	pathname, mode	0(가능), -1(불가)
execve	새 프로그램 실행	path, argv[], envp[]	0(성공), -1(실패)
statfs	파일 시스템 정보 조회	pathname, buf	0 or -1
arch_prctl	x86-64 아키텍처 특정 제어 설정	code, addr	0 or -1
getdents64	디렉터리 엔트리(파일 목록) 읽기	fd, dirp, count	읽은 바이트 수
set_tid_address	쓰레드 ID 저장 주소 설정	tidptr	쓰레드 ID
openat	파일/디렉터리 열기	dirfd, pathname, flags, mode	파일 디스크립터 or -1
newfstatat	파일 상태 정보 조회	dirfd, pathname, statbuf, flags	0 or -1

시스템 콜	역할	주요 인자	반환값 의미
set_robust_list	robust mutex list 설정	head, len	0 or -1
prlimit64	리소스 제한 설정/조회	pid, resource, new, old	0 or -1
getrandom	커널에서 난수 얻기	buf, buflen, flags	생성된 바이트 수
rseq	사용자 공간에서 restartable sequence 등록	struct rseq *, len, flags, sig	0 or -1

시스템 콜 호출 분석

strace의 -c 옵션을 사용해 시스템 콜별 호출 횟수, 총 시간, 평균 시간 등의 정보를 확인할 수 있다.



% time	seconds	usecs/call	calls	errors	syscall
0.00	0.000000	0	5		read
0.00	0.000000	0	9		write
0.00	0.000000	0	9		close
0.00	0.000000	0	18		mmap
0.00	0.000000	0	7		mprotect
0.00	0.000000	0	1		munmap
0.00	0.000000	0	3		brk
0.00	0.000000	0	2		ioctl
0.00	0.000000	0	4		pread64
0.00	0.000000	0	2	2	access
0.00	0.000000	0	1		execve
0.00	0.000000	0	2	2	statfs
0.00	0.000000	0	2	1	arch_prctl
0.00	0.000000	0	2		getdents64
0.00	0.000000	0	1		set_tid_address
0.00	0.000000	0	7		openat
0.00	0.000000	0	8		newfstatat
0.00	0.000000	0	1		set_robust_list
0.00	0.000000	0	1		prlimit64
0.00	0.000000	0	1		getrandom
0.00	0.000000	0	1		rseq
100.00	0.000000	0	87	5	total

ls가 수행되는 동안 mmap 시스템콜이 18번으로 가장 많이 호출되었다.

mmap()은 리눅스에서 파일이나 장치, 메모리 영역을 가상 메모리에 매핑하는 시스템 콜이다.

여러 공유 라이브러리를 로딩하고 locale 데이터를 매핑할 때 mmap()이 사용된다.

뿐만 아니라 ls 실행 중 write()를 위한 내부 버퍼도 mmap으로 할당하기 때문에 가장 많이 사용된 것으로 보인다.

-e 옵션으로 ls 명령어에서 읽는데 성공한 모든 파일의 경로를 확인할 수 있다.

ls가 어떤 파일이나 디렉토리에 접근했는지 추적해 권한 문제나 접근 흔적을 분석할 수 있다.

```
$ strace -e trace=open,openat ls 2>&1 | grep -E 'open|openat' | grep -v '= -1'
```

- strace -e trace=open,openat ls: ls 명령 실행 시 파일 열기 관련 시스템 콜만 추적
- 2>&1: 표준 오류를 표준 출력으로 합침 (strace 출력은 stderr로 나옴)
- grep -E 'open|openat': open 또는 openat만 추출
- grep -v '= -1': 호출 실패 항목 제거 (호출 실패시 반환값은 -1)

위 명령으로 ls가 읽기에 성공한 경로 목록:

/etc/ld.so.cache

/lib/x86_64-linux-gnu/libselinux.so.1

/lib/x86_64-linux-gnu/libc.so.6

/lib/x86_64-linux-gnu/libpcre2-8.so.0

/proc/filesystems

/usr/lib/locale/locale-archive

.

"."은 ls가 실제로 나열하고자 하는 디렉토리이며, 나머지는 실행 준비에 필요한 파일들이다.

5. ls 명령어의 내부 동작 구조 및 원리

ls 명령어는 크게 파일 목록 획득과 파일 목록 출력으로 구분할 수 있으며, 이는 호출된 시스템 콜에 따라 구분된 것이다.

1. 파일 목록 획득

ls가 나열할 파일과 디렉토리의 정보를 커널로부터 얻어오는 작업이 수행된다.

openat() 시스템 콜을 통해 현재 디렉토리를 열고,

getdents64()를 통해 해당 디렉토리의 파일 항목 목록을 가져온다.

이후 각 항목에 대해 fstat() 또는 lstat()를 사용하여 메타데이터를 조회한다.

이 모든 작업은 **유저 공간에서 커널 공간으로 요청하여 파일 시스템으로부터 데이터를 read 하는 과정**이며, ls가 어떤 파일을 사용자에게 보여줄지를 결정하는 부분이다.

2. 파일 목록 출력

파일 목록을 획득하고 이를 사용자에게 보여주기 위해 다시 커널에 write 요청으로 전달된다.

이 과정에서 write() 시스템 콜을 통해 파일 목록이 표준 출력(FD 1번)으로 전달되며, ioctl()이나 isatty()와 같은 시스템 콜은 출력 포맷이나 단말기 속성 확인에 사용된다.

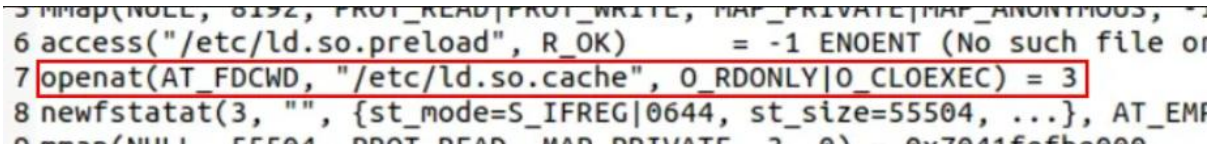
즉, **사용자가 실제로 화면에서 결과를 보게 되는 출력 처리 부분**이다.

이러한 분석을 통해 ls 명령어는 커널과 상호작용하는 시스템 콜들의 흐름을 기반으로 작동함을 확인할 수 있다.

6. 주요 시스템 콜

다음은 ls 명령어뿐만 아니라 많은 명령어에서 자주 사용되는 시스템 콜들이다.

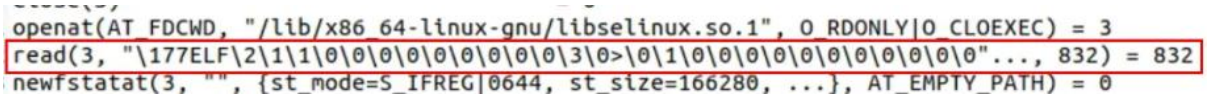
1. openat



지정된 디렉토리를 기준으로 특정 파일을 여는 시스템 콜이다.

위 호출은 현재 작업 디렉터리를 기준으로(AT_FDCWD) 라이브러리 캐시 파일(/etc/ld.so.cache)을 읽기 전용으로(O_RDONLY | O_CLOEXEC) 성공적으로(= 3) 연 것을 의미한다.

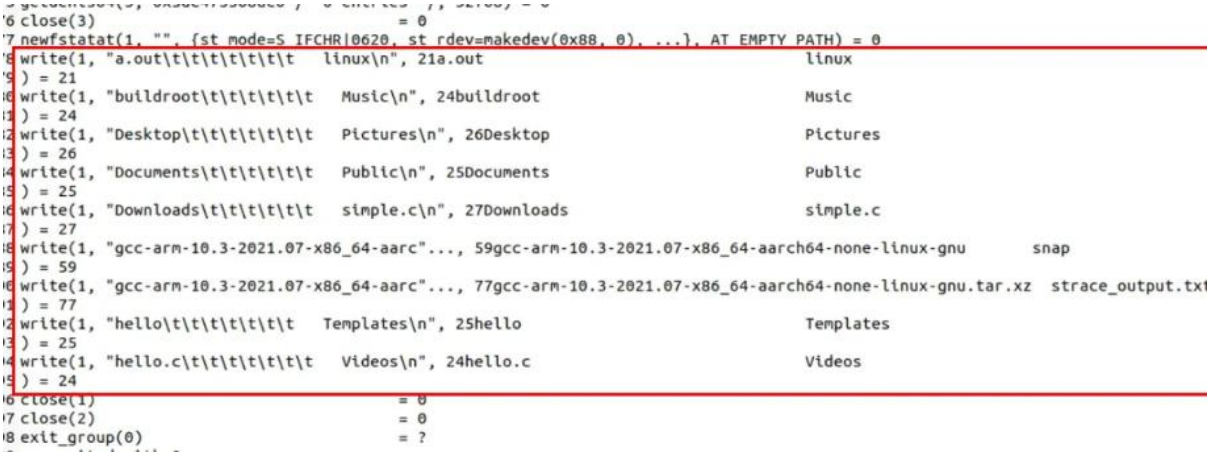
2. read



열린 파일 디스크립터로부터 지정한 바이트 수만큼 데이터를 읽어오는 시스템 콜이다.

위 호출은 파일 디스크립터 3번으로부터 832바이트를 읽어와 버퍼("₩177ELF…")에 저장되어 성공적으로(= 832) 읽혔음을 의미한다.

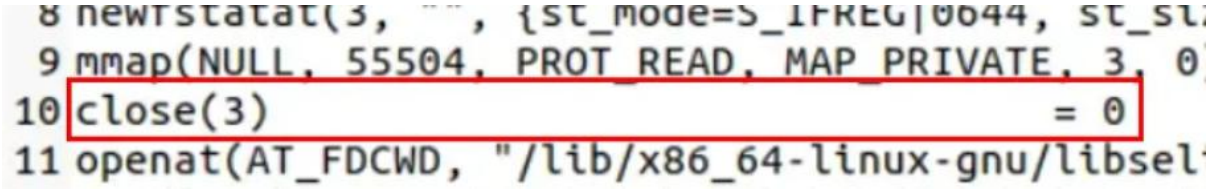
3. write



지정한 파일 디스크립터에 데이터를 출력하는 시스템 콜이다.

위 write 호출들은 파일 디스크립터 1(표준 출력)에 출력할 내용을 성공적으로(= 총 출력 바이트) 출력했음을 의미한다.

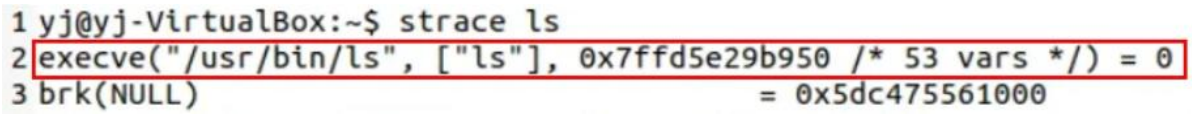
4. close



열린 파일 디스크립터를 닫아 자원을 해제하는 시스템 콜이다.

위 호출은 파일 디스크립터 3번을 닫고, 자원 해제에 성공했음(= 0)을 의미하는 결과이다.

5. execve



지정한 프로그램을 현재 프로세스 공간에서 실행하는 시스템 콜이다.

위 호출은 현재 프로세스에서 "/usr/bin/ls" 실행 파일을 "ls"라는 인자를 전달함과 동시에 새롭게 실행한다는 의미이며, "= 0"은 성공적으로 실행됨을 의미한다.

7. 결론

strace를 통해 ls 명령어의 시스템 콜 호출 과정을 추적함으로써, 단순 명령어조차도 실행 과정에서 수많은 시스템 콜을 통해 커널과 상호작용함을 알 수 있다. 특히 openat, getdents64, write 등은 ls의 핵심 기능과 직결되며, mmap, execve 등은 런타임 환경 구성에 핵심적이다.

이러한 분석은 운영체제 동작 이해, 성능 병목 분석, 시스템 프로그래밍 구현에 매우 유용한 기반이 될 것이다.