| **No.** | **Perintah yang Dijalankan** | **Tujuan / Fungsi** | **Hasil Observasi (Output / Log Penting)** | **Analisis Singkat** |
| --- | --- | --- | --- | --- |
| 1 | strace -e trace=open,read,write,close cat /etc/passwd | Melacak system call yang digunakan saat membuka, membaca, dan menutup file /etc/passwd | - open("/etc/passwd", O\_RDONLY) menunjukkan file dibuka untuk dibaca. - read() menampilkan proses pembacaan isi file. - write() menampilkan hasil yang dikirim ke terminal. - close() menutup file setelah selesai. | strace menunjukkan alur komunikasi antara program dan kernel melalui system call. Menunjukkan bahwa semua operasi file harus melalui kernel. |
| 2 | strace ls | Melihat system call dasar yang terjadi saat menjalankan perintah sederhana. | Banyak system call seperti execve, openat, read, dan write muncul. | Menunjukkan bahwa meskipun sederhana, perintah ls melakukan banyak interaksi dengan kernel. |
| 3 | `dmesg | tail` | Melihat log kernel terbaru setelah menjalankan perintah tertentu. | Menampilkan pesan kernel terakhir, seperti aktivitas driver, mount filesystem, atau error. |
| 4 | `dmesg | grep "error"` | Mendeteksi pesan kesalahan dari kernel. | Menampilkan baris log kernel yang mengandung kata “error”. |

System call penting untuk keamanan sistem operasi karena menjadi **gerbang penghubung antara program pengguna (user space)** dan **kernel (kernel space)**. Berikut alasannya:

1. **Mencegah akses langsung ke kernel**  
   Tanpa system call, program bisa langsung mengakses memori dan sumber daya sistem, yang berpotensi merusak atau mencuri data. System call membatasi akses hanya melalui jalur resmi dan aman.
2. **Menyediakan kontrol dan validasi**  
   Setiap system call diperiksa oleh kernel untuk memastikan permintaan pengguna sah dan tidak berbahaya, misalnya memeriksa izin file atau alamat memori.
3. **Menjaga isolasi antar proses**  
   Dengan system call, setiap proses berjalan terpisah sehingga satu program tidak bisa langsung mengganggu proses lain atau sistem inti.
4. **Mencegah eksekusi kode berbahaya**  
   Kernel dapat menolak operasi yang berpotensi berbahaya (seperti menulis ke area memori kernel), sehingga serangan seperti privilege escalation bisa dicegah.

Mode CPU terpisah

CPU memiliki dua mode: user mode (terbatas) dan kernel mode (akses penuh).

Program biasa berjalan di user mode, sehingga tidak bisa langsung mengakses memori kernel atau perangkat keras.

System call sebagai gerbang resmi

Transisi user → kernel hanya bisa melalui system call atau interrupt yang dikontrol OS.

Kernel memeriksa validitas permintaan sebelum mengeksekusi.

Validasi parameter dan izin

Kernel memeriksa parameter yang dikirim program (misal alamat memori, file, atau proses target).

Jika parameter tidak valid atau operasi tidak diizinkan, kernel menolak permintaan.

Isolasi memori (Memory Protection)

Kernel berada di area memori yang terisolasi dari user space.

User program tidak bisa membaca atau menulis langsung ke memori kernel.

Interrupt dan trap handling

Semua system call memicu software interrupt/trap, yang menandai transisi ke kernel.

Kernel menangani interrupt secara aman dan mengembalikan kontrol ke user mode setelah selesai.

File dan I/O

open() → membuka file

read() → membaca data dari file

write() → menulis data ke file

close() → menutup file

lseek() → memindahkan posisi baca/tulis dalam file

Proses dan manajemen

fork() → membuat proses baru

exec() → menjalankan program baru dalam proses

wait() → menunggu proses anak selesai

exit() → mengakhiri proses

Memori

mmap() → memetakan file atau perangkat ke memori

brk() → mengatur ukuran heap proses

Kontrol akses dan keamanan

chmod() → mengubah hak akses file

chown() → mengubah pemilik file

getuid() / geteuid() → mendapatkan ID pengguna

Interaksi dengan perangkat dan sistem

ioctl() → mengontrol perangkat keras atau driver

stat() → mendapatkan informasi file

pipe() → membuat saluran komunikasi antar proses