**NAMA:BACHTIAR DWI INDRIANTO**

**NIM:** **250320581**

**KELAS:1DSRA**

**JURUSAN:SAINS DATA**

**1. Mengapa *System Call* Penting untuk Keamanan Sistem Operasi**

*System call* bisa dibilang adalah “jembatan” antara program yang kita jalankan dan inti dari sistem operasi (kernel). Semua aplikasi di komputer — seperti browser, pemutar musik, atau editor teks — berjalan di ruang pengguna (*user space*), yang memiliki hak akses terbatas. Sementara kernel memiliki kendali penuh atas semua hal penting, seperti memori, file, dan perangkat keras.

Nah, agar aplikasi bisa melakukan hal-hal seperti membuka file, mengirim data ke internet, atau mencetak dokumen, mereka harus “meminta izin” ke kernel lewat *system call*. Jadi, *system call* berfungsi sebagai pintu masuk yang aman. Tanpa mekanisme ini, program bisa langsung mengakses perangkat keras sesuka hati — dan itu sangat berbahaya, karena bisa menyebabkan kebocoran data, kerusakan sistem, atau bahkan diambil alih oleh malware.

Selain itu, *system call* membantu sistem tetap stabil. Kalau sebuah aplikasi rusak atau melakukan kesalahan, efeknya hanya terjadi di ruang pengguna. Kernel tetap aman karena program tidak punya izin langsung untuk mengubahnya. Dengan kata lain, *system call* menjaga agar program tidak “seenaknya” mengubah bagian inti sistem.

**2. Bagaimana OS Menjaga Agar Transisi User–Kernel Tetap Aman**

Ketika program memanggil *system call*, sistem operasi harus berpindah dari *user mode* ke *kernel mode*. Perpindahan ini sangat sensitif, karena kalau ada celah keamanan di proses ini, penyerang bisa saja mengambil alih kendali kernel.

Untuk mencegah hal itu, sistem operasi dan prosesor bekerja sama. Prosesor punya lapisan perlindungan yang disebut *privilege rings*. Biasanya, program biasa berjalan di *ring 3* (paling rendah), sedangkan kernel berada di *ring 0* (paling tinggi). Program di *ring 3* tidak bisa langsung menjalankan perintah khusus yang bisa memengaruhi perangkat keras atau memori kernel.

Ketika sebuah *system call* terjadi, prosesor mengaktifkan mekanisme khusus seperti syscall (atau int 0x80 pada sistem lama). Kernel kemudian memeriksa data yang dikirim oleh program — apakah parameter valid, apakah pengguna punya izin yang cukup, dan apakah alamat memori yang dipakai aman. Kalau ada yang mencurigakan, kernel akan menolak permintaan tersebut dan memberikan pesan kesalahan.

Setelah tugas selesai, kontrol dikembalikan ke *user mode* dengan cara yang hati-hati agar tidak ada data sensitif kernel yang terbawa keluar. Semua langkah ini membuat proses transisi antara *user mode* dan *kernel mode* tetap aman dan terkendali.

**3. Contoh *System Call* yang Sering Digunakan di Linux**

Ada ratusan *system call* di Linux, tapi beberapa yang paling sering digunakan antara lain:

* **read()** – untuk membaca data dari file atau input seperti keyboard.
* **write()** – untuk menulis data ke file, layar, atau jaringan.
* **open()** – untuk membuka file dan mendapatkan akses ke dalamnya.
* **close()** – untuk menutup file yang sudah selesai digunakan.
* **fork()** – untuk membuat proses baru dari proses yang sudah ada.
* **exec()** – untuk menjalankan program baru.
* **wait()** – untuk menunggu proses anak selesai.
* **exit()** – untuk keluar dari program.
* **mmap()** – untuk memetakan file ke dalam memori agar lebih cepat diakses.

Semua *system call* itu adalah cara aman bagi program untuk “berbicara” dengan kernel. Tanpa *system call*, tidak ada batas yang jelas antara aplikasi dan inti sistem, dan komputer akan jauh lebih rentan terhadap kesalahan dan serangan.