

PERTEMUAN 2

STRUKTUR SISTEM OPERASI & SYSTEM CALL

Mata Kuliah	Sistem Operasi (SI-204) — 3 SKS (2 Teori, 1 Praktikum)
Pertemuan	2 dari 16
Sub-CPMK	Sub-CPMK-2: Mampu membandingkan struktur sistem operasi dan menjelaskan mekanisme system call (C2–C4).
CPMK Terkait	CPMK-1 (didukung praktikum untuk CPMK-4)
Indikator	• Menjelaskan alur system call • Membandingkan struktur monolitik, berlapis, mikrokernel, dan hibrida • Menjelaskan proses boot
Bentuk & Waktu	Studi kasus perbandingan Linux, Windows, dan macOS (TM 2×50'); Praktikum (1×170'); Penugasan Terstruktur (2×60'); Belajar Mandiri (2×60')
Penilaian	Latihan Kasus 1 dan Laporan Praktikum 2 (bobot 2% dari nilai akhir, bagian komponen Latihan Kasus & Laporan Praktikum)
Referensi	[1] Bab 2; [3] Bab 2; [2] Bab 1; [5]
Prasyarat	Materi Pertemuan 1 (dual-mode, trap, interupsi) dan VM Ubuntu Server dari Praktikum 1 dengan OpenSSH terpasang

A. TUJUAN PEMBELAJARAN

Setelah mengikuti pertemuan ini, mahasiswa diharapkan mampu:

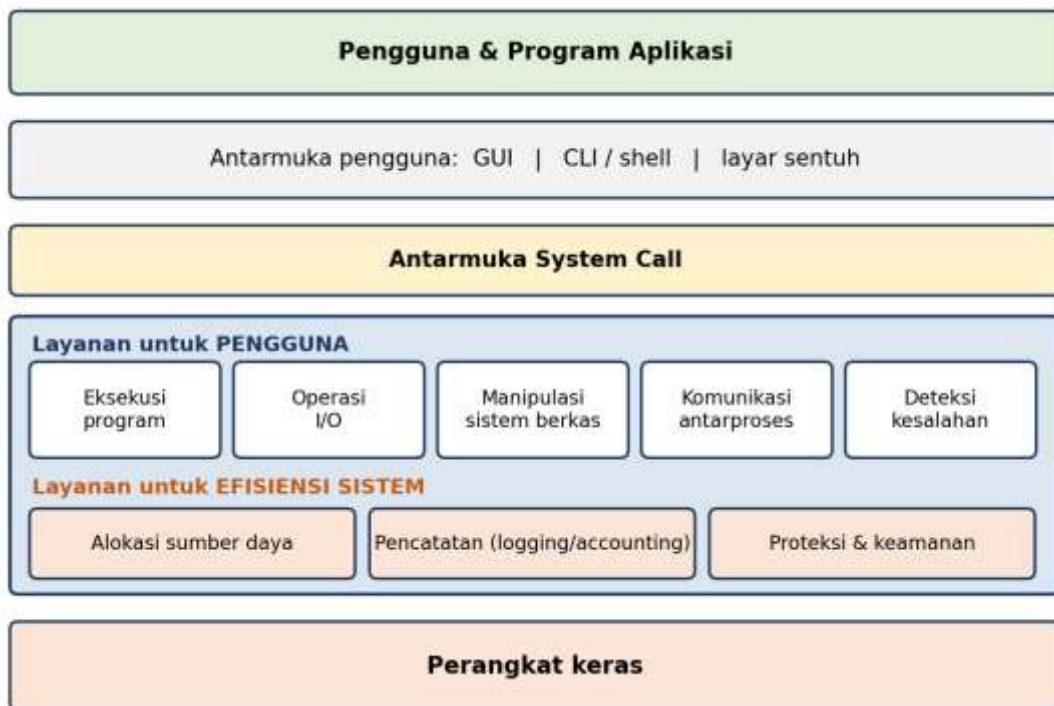
1. Menjelaskan layanan yang disediakan sistem operasi bagi pengguna dan bagi efisiensi sistem.
2. Menjelaskan perbedaan API dan system call serta menguraikan alur lengkap sebuah system call.
3. Mengelompokkan system call berdasarkan jenisnya dan memberi contoh pada Linux dan Windows.
4. Membandingkan struktur monolitik, berlapis, mikrokernel, modular, dan hibrida dari sisi kinerja, keandalan, dan keamanan.
5. Menjelaskan tahapan proses boot dari firmware hingga layanan siap melayani pengguna.
6. (Praktikum) Mengakses server melalui SSH, mengenali hierarki direktori Linux, mengelola pengguna dan paket, serta mengamati system call menggunakan `strace`.

B. LAYANAN SISTEM OPERASI

B.1 Mengingat Kembali

Pada Pertemuan 1 kita mengenal OS sebagai "pengelola gedung" yang berdiri di antara aplikasi dan perangkat keras. Pertemuan ini menjawab tiga pertanyaan lanjutan: **layanan apa saja** yang disediakan pengelola itu, **bagaimana cara aplikasi memesan layanan** tersebut (system call), dan **bagaimana "kantor" pengelola ditata di dalamnya** (struktur OS).

B.2 Dua Kelompok Layanan



Gambar 2.1 Layanan sistem operasi bagi pengguna dan bagi efisiensi sistem

Layanan	Penjelasan	Contoh
Antarmuka pengguna	Cara pengguna berinteraksi dengan OS: GUI, CLI (<i>shell</i>), atau layar sentuh.	Desktop Windows, terminal <i>bash</i> , layar Android
Eksekusi program	Memuat program ke memori, menjalankannya, dan mengakhirinya, baik normal maupun karena kesalahan.	Menjalankan <i>python3 app.py</i>
Operasi I/O	Program tidak mengakses perangkat secara langsung. OS yang melakukannya atas nama program.	Membaca data dari kartu jaringan
Manipulasi sistem berkas	Membuat, membaca, menulis, menghapus, dan mencari berkas serta direktori, termasuk mengatur izinnya.	Menyimpan berkas unggahan tugas di LMS
Komunikasi	Pertukaran data antarproses di satu komputer atau antarkomputer di jaringan (<i>shared memory</i> atau <i>message passing</i>).	Aplikasi web mengirim query ke DBMS
Deteksi kesalahan	Mendeteksi dan menangani kesalahan pada CPU, memori, perangkat, atau program.	Disk penuh, koneksi terputus, <i>segmentation fault</i>
Alokasi sumber daya	Membagi CPU, memori, dan perangkat ketika banyak proses berjalan bersamaan.	Penjadwalan CPU (Pert. 4)
Pencatatan (<i>logging/accounting</i>)	Mencatat siapa memakai sumber daya apa dan kejadian penting di sistem.	<i>journalctl</i> , <i>/var/log/auth.log</i>
Proteksi & keamanan	Mengendalikan akses ke sumber daya dan melindungi sistem dari pihak luar.	Login, izin berkas, <i>firewall</i>

Konteks SI

Layanan **pencatatan** sering diremehkan, padahal log adalah sumber utama ketika tim TI menyelidiki insiden, misalnya siapa yang login ke server basis data pukul 02.00 dini hari. Audit sistem informasi dan kepatuhan terhadap regulasi perlindungan data juga bergantung pada log yang lengkap.

B.3 Antarmuka Pengguna: CLI vs GUI

Command-Line Interface (CLI) atau **shell** menerima perintah berbentuk teks. Contohnya **bash** dan **zsh** di Linux/macOS serta PowerShell di Windows. **Graphical User Interface (GUI)** memakai jendela, ikon, dan menu. Perlu dicatat bahwa shell **bukan bagian dari kernel**. Shell hanyalah program biasa di user mode yang membaca perintah Anda lalu meminta kernel menjalankannya melalui system call.

Aspek	CLI (shell)	GUI
Kemudahan belajar	Perlu menghafal perintah	Intuitif bagi pemula
Otomatisasi	Sangat mudah dengan <i>script</i>	Sulit, umumnya manual
Sumber daya	Sangat ringan	Butuh memori dan grafis lebih besar
Akses jarak jauh	Ringan melalui SSH	Butuh <i>remote desktop</i> yang lebih berat
Pemakaian umum	Server, administrasi sistem, DevOps	Komputer pengguna akhir

Karena itu hampir semua server layanan SI dikelola melalui CLI, termasuk VM Ubuntu Server yang Anda pasang pada Praktikum 1.

C. SYSTEM CALL

C.1 Apa Itu System Call?

System call adalah **antarmuka resmi antara program dan kernel**. Setiap kali program membutuhkan sesuatu yang hanya boleh dikerjakan kernel, seperti membuka berkas, membuat proses baru, atau mengirim data lewat jaringan, program harus memanggil system call. Ingat kembali Pertemuan 1: pemanggilan ini memicu **trap** yang memindahkan CPU dari user mode ke kernel mode.

Analogi: Loker Layanan Bank

Nasabah tidak boleh masuk ke ruang brankas. Untuk menarik uang, nasabah mengisi formulir di **loket** (system call). Petugas memeriksa identitas dan saldo (validasi oleh kernel), mengambil uang dari brankas (akses perangkat keras), lalu menyerahkannya kepada nasabah (hasil dikembalikan ke program). Jenis formulir berbeda untuk setiap layanan: tarik tunai, transfer, cetak buku. Begitu pula system call: ada **read()**, **write()**, **fork()**, dan ratusan lainnya.

C.2 Berapa Banyak System Call dalam Satu Perintah Sederhana?

Ambil contoh perintah **cp nilai.csv backup.csv** untuk menyalin berkas. Di balik perintah sederhana itu, program menjalankan urutan system call kira-kira seperti berikut:

1. Membuka berkas sumber **nilai.csv** → **open()** / **openat()**. Jika berkas tidak ada atau tidak boleh dibaca, program menampilkan pesan kesalahan dan berhenti.
2. Membuat berkas tujuan **backup.csv** → **openat()** dengan opsi **create**. Jika sudah ada, program mungkin bertanya apakah akan menyimpannya.

3. Mengulang: membaca sepotong data dari sumber → `read()`, lalu menulisnya ke tujuan → `write()`, sampai seluruh isi berkas tersalin.
4. Menutup kedua berkas → `close()`.
5. Mengakhiri program → `exit_group()`.

Belum termasuk system call untuk memuat program `cp` itu sendiri ke memori. Pada praktikum Anda akan melihat bahwa perintah sederhana `ls` bisa memicu puluhan hingga lebih dari seratus system call. Jadi program **selalu** bergantung pada kernel, meskipun pemrogramnya jarang menyadarinya.

C.3 API vs System Call

Pemrogram hampir tidak pernah memanggil system call secara langsung. Mereka memakai **Application Programming Interface (API)**, yaitu kumpulan fungsi pustaka yang lebih mudah dipakai. Fungsi API inilah yang di dalamnya memanggil system call.

Aspek	API	System call
Letak	Pustaka di user mode (mis. glibc, Win32 API)	Pintu masuk ke kernel
Contoh	<code>printf()</code> , <code>fopen()</code> , <code>open()</code> pada Python, <code>CreateFile()</code>	<code>write</code> , <code>openat</code> , <code>NtCreateFile</code>
Hubungan	Satu fungsi API bisa memanggil nol, satu, atau banyak system call	Dipanggil oleh pustaka melalui instruksi khusus CPU
Stabilitas	Dijaga stabil antarversi OS	Bisa berubah antarversi (terutama di Windows)
Portabilitas	Standar POSIX memungkinkan program yang sama dikompilasi di Linux, macOS, dan UNIX lain	Spesifik untuk tiap OS dan arsitektur CPU

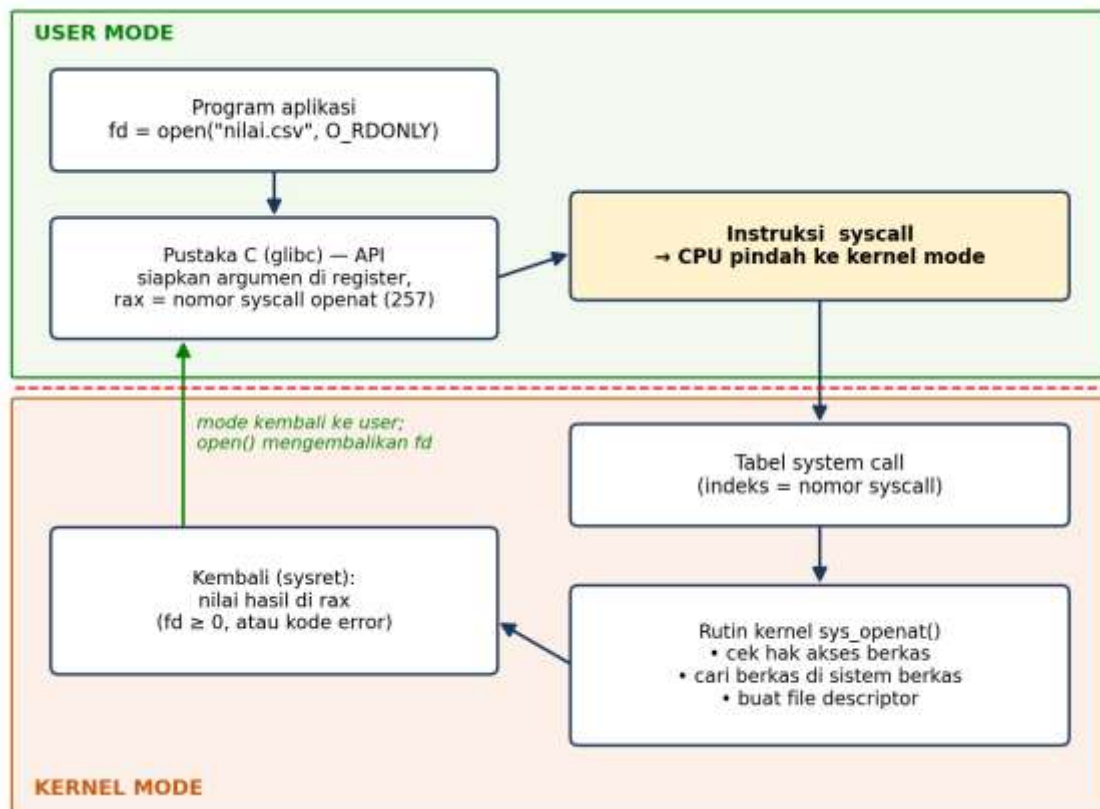
Tiga API yang paling dikenal: **POSIX API** (Linux, macOS, dan UNIX lain), **Windows API/Win32**, dan **Java API** yang berjalan di atas JVM. Mengapa memakai API? Karena **portabel**: program yang ditulis dengan API standar dapat berjalan di banyak sistem tanpa perlu diubah. API juga lebih mudah dipakai daripada system call mentah yang menuntut detail seperti nomor syscall dan isi register.

Luruskan Miskonsepsi

`printf()` **bukan** system call. `printf()` adalah fungsi pustaka C yang memformat teks lalu menaruhnya di *buffer*. Baru ketika buffer penuh atau program berakhir, pustaka memanggil system call `write()`. Contoh sebaliknya: fungsi `strlen()` untuk menghitung panjang string tidak memanggil system call sama sekali, karena tidak ada yang perlu dikerjakan kernel.

C.4 Alur Lengkap Sebuah System Call

Gambar 2.2 memperlihatkan apa yang terjadi ketika program di Linux (CPU x86-64) memanggil `open()` untuk membuka berkas.



Gambar 2.2 Alur system call `open()` pada Linux x86-64

1. Program memanggil fungsi pustaka `open()`.
2. Pustaka C menaruh **nomor system call** ke register `rax` (nomor untuk `openat` adalah 257 pada x86-64) dan menaruh argumen (nama berkas, mode) ke register lain.
3. Pustaka menjalankan instruksi `syscall`. CPU menyimpan posisi program, berpindah ke **kernel mode**, dan melompat ke titik masuk system call di kernel.
4. Kernel memakai nomor di `rax` sebagai indeks ke **tabel system call** untuk menemukan rutin yang tepat, yaitu `sys_openat`.
5. Rutin kernel **memvalidasi** argumen dan hak akses, lalu mengerjakan tugasnya.
6. Hasil (nomor *file descriptor* atau kode kesalahan) ditaruh di `rax`, lalu CPU **kembali ke user mode** dan program melanjutkan eksekusi.

Catatan: instruksi dan register yang dipakai berbeda antararsitektur (misalnya `svc` pada ARM64), tetapi polanya selalu sama: **nomor layanan + argumen** → **trap ke kernel** → **validasi** → **kerjakan** → **kembali**.

C.5 Cara Mengirim Parameter ke Kernel

- **Lewat register:** paling cepat, dipakai ketika jumlah argumen sedikit. Linux x86-64 memakai hingga enam register untuk argumen.
- **Lewat blok/tabel di memori:** argumen disimpan di memori, lalu alamat blok tersebut dikirim lewat register. Cara ini dipakai untuk data yang besar, misalnya struktur berisi banyak kolom.
- **Lewat stack:** program mendorong (*push*) argumen ke stack dan kernel mengambilnya (*pop*).

Apa pun caranya, kernel **tidak boleh memercayai** alamat yang dikirim program begitu saja. Kernel harus memeriksa bahwa alamat tersebut memang milik proses pemanggil. Tanpa pemeriksaan ini, program jahat dapat menipu kernel agar membaca atau menulis memori kernel.

C.6 Jenis-jenis System Call

Jenis	Fungsi	Windows	Linux / UNIX
Kontrol proses	Membuat, mengakhiri, dan menunggu proses	CreateProcess(), ExitProcess(), WaitForSingleObject()	fork(), exec(), exit(), wait()
Manajemen berkas	Membuat, membuka, membaca, menulis, dan menutup berkas	CreateFile(), ReadFile(), WriteFile(), CloseHandle()	open(), read(), write(), close()
Manajemen perangkat	Meminta, melepas, dan mengatur perangkat	SetConsoleMode(), ReadConsole(), WriteConsole()	ioctl(), read(), write()
Pemeliharaan informasi	Membaca atau mengatur waktu, data sistem, dan atribut proses	GetCurrentProcessID(), SetTimer(), Sleep()	getpid(), alarm(), sleep()
Komunikasi	Pertukaran data antarproses atau antarkomputer	CreatePipe(), CreateFileMapping(), MapViewOfFile()	pipe(), shm_open(), mmap(), socket()
Proteksi	Mengatur izin akses sumber daya	SetFileSecurity(), InitializeSecurityDesc riptor()	chmod(), umask(), chown()

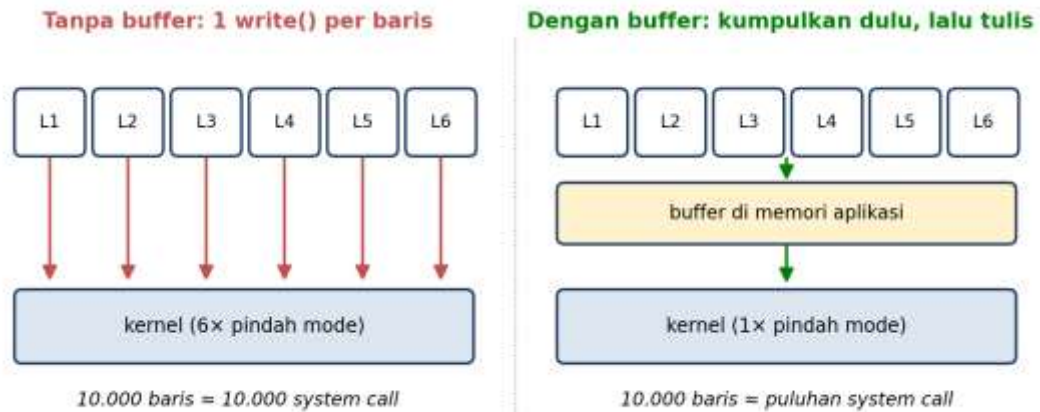
Sumber pengelompokan dan contoh: Silberschatz dkk. [1], Bab 2. Nama di kolom Windows adalah fungsi Win32 API yang di baliknyanya memanggil system call internal Windows.

Tahukah Anda?

Di Linux, daftar dan nomor system call bersifat **terdokumentasi dan dijaga stabil**, sehingga program lama tetap berjalan di kernel baru. Di Windows, system call internal (misalnya `NtCreateFile` di `ntdll.dll`) **tidak didokumentasikan untuk umum** dan nomornya dapat berubah antarversi. Program Windows dianjurkan selalu memakai Win32 API. Perbedaan ini menjelaskan mengapa program Linux tidak bisa langsung dijalankan di Windows, dan sebaliknya, walaupun CPU-nya sama.

C.7 System Call Tidak Gratis: Dampaknya bagi Kinerja Aplikasi SI

Setiap system call melibatkan perpindahan mode, validasi, dan penyimpanan/pemulihan state. Biaya satu system call memang kecil, tetapi jika dipanggil ribuan hingga jutaan kali, totalnya menjadi signifikan. Karena itu pustaka memakai **buffering**: data dikumpulkan dulu di memori aplikasi, lalu dikirim ke kernel sekaligus dalam satu system call (Gambar 2.3).



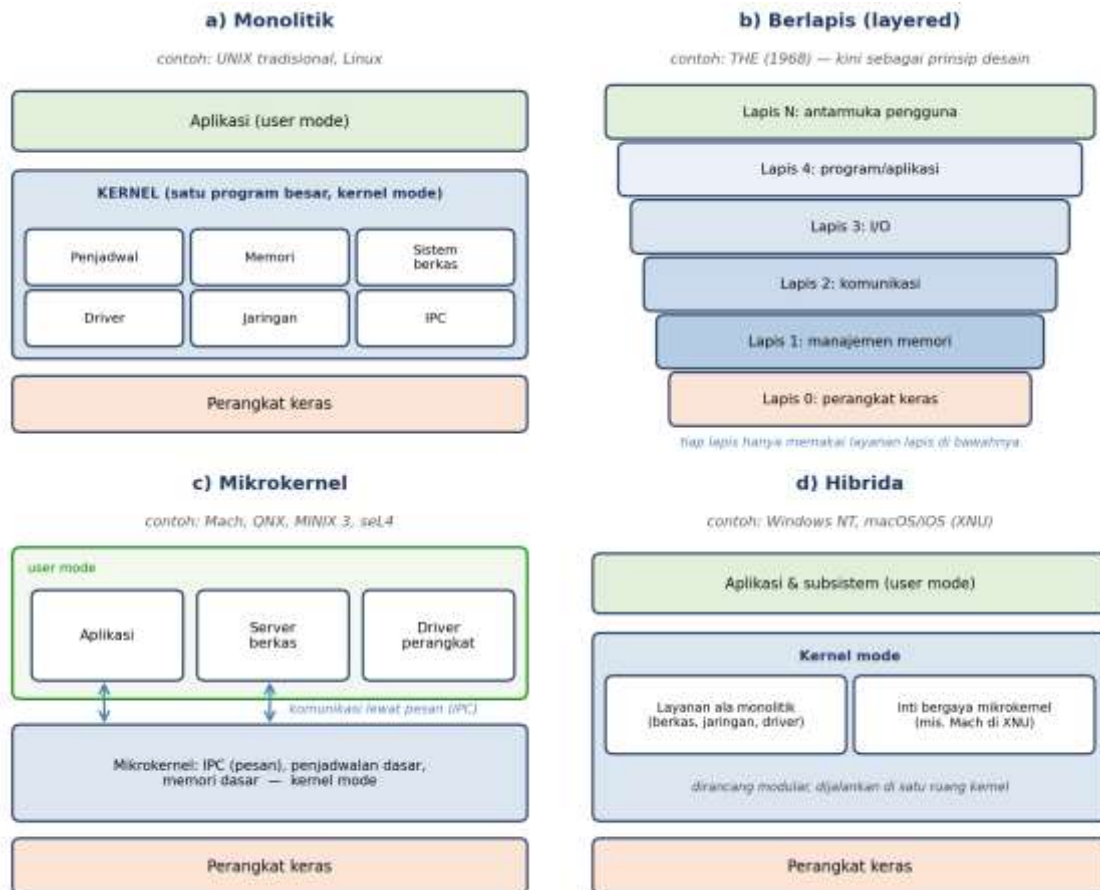
Gambar 2.3 Menulis log tanpa dan dengan buffer

Konteks SI

Aplikasi yang menulis log transaksi **baris per baris langsung ke kernel** akan jauh lebih lambat dibanding aplikasi yang memakai buffer. Namun buffering punya risiko: jika server mati mendadak, data yang masih di buffer dan belum ditulis ke disk bisa hilang. Karena itu DBMS memanggil `fsync()` saat transaksi di-*commit* untuk memastikan data benar-benar tersimpan. Inilah *trade-off* klasik antara **kinerja** dan **ketahanan data**. Anda akan mengukurnya sendiri pada Praktikum 2 Langkah 6.

D. STRUKTUR SISTEM OPERASI

Kernel OS modern berisi jutaan baris kode. Cara kode itu **dibagi dan ditata** menentukan kecepatan, keandalan, keamanan, dan kemudahan pengembangan OS. Ada empat pendekatan utama ditambah satu teknik modular yang dipakai hampir semua OS modern (Gambar 2.4).



Gambar 2.4 Empat pendekatan struktur sistem operasi

D.1 Struktur Monolitik

Seluruh layanan kernel (penjadwalan, memori, sistem berkas, driver, jaringan) dikemas dalam **satu program besar** yang berjalan di kernel mode dan berbagi satu ruang alamat. UNIX tradisional dan Linux memakai pendekatan ini.

- **Kelebihan:** sangat cepat, karena komponen kernel saling memanggil sebagai fungsi biasa tanpa perlu berkirir pesan atau berpindah mode.
- **Kekurangan:** satu *bug* di satu komponen, misalnya driver kartu grafis, dapat merusak seluruh kernel dan menjatuhkan sistem (*kernel panic*). Kode yang besar juga lebih sulit dipelihara dan diuji.

D.2 Struktur Berlapis (Layered)

OS dibagi menjadi beberapa lapisan. Lapis 0 adalah perangkat keras, lapis tertinggi adalah antarmuka pengguna, dan **setiap lapis hanya boleh memakai layanan lapis di bawahnya**. Contoh klasiknya OS THE (Dijkstra, 1968).

- **Kelebihan:** mudah dirancang dan di-*debug* karena setiap lapis dapat diuji satu per satu dari bawah.
- **Kekurangan:** sulit menentukan urutan lapis yang tepat, karena banyak komponen saling membutuhkan. Setiap permintaan juga harus melewati banyak lapis sehingga lebih lambat.

Kini OS murni berlapis jarang dipakai. Prinsip berlapis tetap hidup sebagai **prinsip desain** di dalam komponen OS, misalnya tumpukan protokol jaringan TCP/IP.

D.3 Struktur Mikrokernel

Kernel dibuat **sekecil mungkin**, hanya berisi fungsi paling dasar: komunikasi antarproses (IPC), penjadwalan dasar, dan manajemen memori dasar. Layanan lain seperti sistem berkas dan **driver perangkat dipindahkan ke user mode** sebagai proses biasa yang disebut *server*. Komponen-komponen itu berkomunikasi dengan **mengirim pesan** melalui mikrokernel. Contohnya Mach, QNX, MINIX 3, dan seL4.

- **Kelebihan:** lebih andal dan aman. Jika driver *crash*, yang mati hanya proses driver itu dan dapat dijalankan ulang tanpa menjatuhkan sistem. Kernel yang kecil juga lebih mudah diverifikasi; seL4, misalnya, telah dibuktikan kebenarannya secara matematis (*formal verification*).
- **Kekurangan:** lebih lambat, karena layanan yang dulunya pemanggilan fungsi biasa kini menjadi pengiriman pesan antarproses yang melibatkan perpindahan mode berulang kali.

Mikrokernel banyak dipakai di sistem yang **menuntut keandalan tinggi**, seperti sistem kendali kendaraan dan perangkat industri. QNX adalah contoh yang banyak dipakai di industri otomotif.

D.4 Modul Kernel yang Dapat Dimuat (Loadable Kernel Modules)

Hampir semua OS modern memakai **modul**: bagian kernel (terutama driver dan sistem berkas) yang dapat **dimuat dan dilepas saat sistem berjalan** tanpa harus mengompilasi ulang atau *restart* kernel. Linux adalah contoh utama. Ketika Anda mencolokkan *flashdisk*, kernel dapat memuat modul driver USB yang sesuai secara otomatis.

Penting dipahami: modul Linux tetap berjalan **di kernel mode dan di ruang alamat kernel**. Jadi Linux tetap **monolitik**, hanya saja **modular**. Modul membuat kernel lebih fleksibel, tetapi tidak menambah isolasi. Modul yang *buggy* tetap dapat menjatuhkan seluruh sistem.

D.5 Struktur Hibrida

Dalam praktik, sebagian besar OS populer **menggabungkan** beberapa pendekatan demi menyeimbangkan kinerja, keamanan, dan kemudahan pengembangan.

- **Windows (NT):** dirancang dengan konsep modular ala mikrokernel, tetapi demi kinerja sebagian besar layanan (manajer memori, sistem berkas, driver, grafis) dijalankan di kernel mode. Aplikasi berinteraksi melalui *subsystem* dan Win32 API.
- **macOS dan iOS (kernel XNU):** menggabungkan mikrokernel **Mach** (IPC, penjadwalan, memori) dengan komponen **BSD** (proses, jaringan, sistem berkas, API POSIX) serta kerangka driver I/O Kit, semuanya berjalan di satu ruang kernel.
- **Android:** memakai kernel **Linux** (monolitik-modular) ditambah lapisan HAL (*Hardware Abstraction Layer*), runtime ART, dan *framework* aplikasi di user mode.

D.6 Perbandingan Struktur

Aspek	Monolitik	Berlapis	Mikrokernel	Hibrida
Kinerja	Sangat tinggi	Rendah–sedang	Sedang (overhead IPC)	Tinggi
Isolasi kesalahan	Rendah (bug driver menjatuhkan sistem)	Sedang	Tinggi (driver di user mode)	Sedang
Ukuran kode kernel	Besar	Besar	Kecil	Besar
Kemudahan verifikasi	Sulit	Relatif mudah per lapis	Paling mudah	Sulit

Aspek	Monolitik	Berlapis	Mikrokernel	Hibrida
Contoh	Linux, UNIX tradisional	THE	QNX, MINIX 3, seL4	Windows NT, macOS/iOS

Label pada tabel adalah kecenderungan umum, bukan ukuran pasti. Kinerja nyata sangat bergantung pada implementasinya. Mikrokernel modern seperti seL4 telah memangkas overhead IPC secara drastis.

Prinsip Desain: Pisahkan Mekanisme dan Kebijakan

Mekanisme menjawab *bagaimana* sesuatu dilakukan, sedangkan **kebijakan** (*policy*) menjawab *apa* yang dilakukan. Contohnya timer (Pertemuan 1) adalah mekanisme untuk merebut CPU, sedangkan berapa lama jatah CPU tiap proses adalah kebijakan. OS yang baik memisahkan keduanya, sehingga kebijakan dapat diubah (misalnya prioritas layanan basis data dinaikkan) tanpa mengubah mekanismenya. Anda akan melihat prinsip ini lagi pada penjadwalan CPU (Pertemuan 4).

E. PROSES BOOT

Bootting adalah proses menyalakan komputer hingga kernel berjalan dan sistem siap dipakai. Masalahnya, saat listrik baru mengalir, RAM masih kosong, padahal kernel tersimpan di disk. Siapa yang memuat kernel? Jawabannya adalah rantai program kecil yang masing-masing memuat program berikutnya (Gambar 2.5).



Gambar 2.5 Tahapan boot pada sistem Linux modern

Tahap	Penjelasan
Firmware	Program yang tersimpan di chip <i>flash</i> pada motherboard. BIOS adalah firmware lama yang membaca 512 byte pertama disk (MBR). UEFI adalah penggantinya pada komputer modern: mendukung disk lebih dari 2 TB (tabel partisi GPT), memuat bootloader dari partisi khusus bernama <i>EFI System Partition</i> , dan mendukung Secure Boot .
POST	<i>Power-On Self-Test</i> : pemeriksaan dasar CPU, RAM, dan perangkat. Jika gagal, biasanya muncul bunyi <i>beep</i> atau pesan kesalahan.
Bootloader	Program yang memuat kernel. Di Linux umumnya GRUB , yang juga menampilkan menu jika ada beberapa OS atau beberapa versi kernel.
Kernel & initramfs	GRUB memuat berkas kernel (vmlinux) dan initramfs , yaitu sistem berkas kecil sementara di RAM berisi driver yang dibutuhkan untuk menemukan disk utama.
Inisialisasi kernel	Kernel menyiapkan CPU, memori, interupsi, dan driver, lalu memasang (<i>mount</i>) sistem berkas root dari disk.
PID 1 (init)	Kernel menjalankan proses pertama di user mode, di Ubuntu adalah systemd . Semua proses lain di sistem adalah "keturunan" proses ini.
Layanan & target	systemd menyalakan layanan secara paralel sesuai ketergantungannya (jaringan, SSH, basis data, web) hingga mencapai <i>target</i> , misalnya multi-user.target untuk server tanpa GUI.

Secure Boot

Dengan Secure Boot, firmware UEFI hanya menjalankan bootloader yang **ditandatangani secara digital** oleh pihak terpercaya, dan bootloader hanya memuat kernel yang juga ditandatangani. Tujuannya mencegah *bootkit*, yaitu malware yang menyusup sebelum OS berjalan sehingga sulit dideteksi antivirus.

Konteks SI

Waktu boot server menentukan **berapa lama layanan SI mati** setelah *restart* karena pembaruan keamanan atau pemadaman listrik. Administrator dapat menganalisis layanan mana yang memperlambat boot dengan **systemd-analyze blame**, dan memastikan layanan penting (basis data, web) diatur untuk **menyala otomatis** saat boot dengan **systemctl enable**. Anda akan mencobanya di praktikum.

F. STUDI KASUS: MEMBANDINGKAN LINUX, WINDOWS, DAN macOS

F.1 Profil Struktur Tiga OS

Aspek	Linux (mis. Ubuntu)	Windows 11 / Server	macOS
Struktur kernel	Monolitik-modular	Hibrida (NT)	Hibrida (XNU: Mach + BSD)
API utama	POSIX / glibc	Win32 API, .NET	POSIX + framework Apple (Cocoa)
System call	Terdokumentasi, stabil	Internal, tidak didokumentasikan untuk umum	Ada, tetapi aplikasi dianjurkan memakai pustaka sistem
Driver	Modul kernel (banyak ikut dalam kode sumber kernel)	Driver dari vendor, ditandatangani Microsoft	Semakin didorong berjalan di user mode (DriverKit)

Aspek	Linux (mis. Ubuntu)	Windows 11 / Server	macOS
Lisensi	Open source (GPL)	Proprietary	Proprietary (sebagian komponen open source)
Pemakaian dominan di SI	Server web, basis data, cloud, kontainer	Desktop kantor, Active Directory, aplikasi .NET	Laptop pengembang dan desain

F.2 Deskripsi Kasus

Bagian TI **Universitas Nusantara** (fiktif) mengelola tiga jenis perangkat: server SIAKAD berbasis Ubuntu Server, server Active Directory berbasis Windows Server, dan laptop tim pengembang yang sebagian besar macOS. Dalam satu bulan terjadi tiga peristiwa:

- Server SIAKAD mengalami *kernel panic* setelah administrator memasang driver kartu jaringan dari situs pihak ketiga yang tidak resmi.
- Seorang pengembang mencoba menjalankan berkas program hasil kompilasi Linux langsung di laptop Windows, tetapi gagal. Setelah memakai WSL2, program tersebut dapat berjalan.
- Vendor menawarkan sistem antrean layanan mahasiswa berbasis perangkat kiosk yang memakai OS mikrokernel dan mengklaim "driver layar sentuh yang crash tidak akan mematikan kiosk".

F.3 Pertanyaan Diskusi

- Mengapa driver yang bermasalah dapat menjatuhkan seluruh server Ubuntu? Bagaimana struktur kernel Linux menjelaskan hal ini?
- Mengapa program Linux tidak dapat langsung berjalan di Windows, padahal CPU-nya sama-sama x86-64? Mengapa WSL2 menyelesaikan masalah ini?
- Apakah klaim vendor kiosk masuk akal? Apa *trade-off* yang harus diterima?
- Rekomendasi apa yang Anda berikan kepada Bagian TI agar peristiwa (a) tidak terulang?

F.4 Pembahasan

- Driver berjalan di kernel mode.** Linux monolitik-modular: modul driver dimuat ke ruang alamat kernel dan berjalan di kernel mode. Tidak ada lapisan pengawas di atasnya, sehingga bug di driver dapat merusak struktur data kernel dan kernel memilih berhenti total (*kernel panic*) demi mencegah kerusakan data yang lebih parah.
- System call dan format program berbeda.** Program hasil kompilasi Linux memanggil system call Linux (dengan nomor dan aturan Linux) dan tersimpan dalam format berkas ELF. Windows memakai format PE dan system call yang sama sekali berbeda, sehingga kernel Windows tidak memahami permintaan program tersebut. WSL2 menjalankan **kernel Linux sungguhan di dalam mesin virtual ringan**, sehingga system call Linux dilayani oleh kernel Linux. Konsep virtualisasi ini dibahas pada Pertemuan 14.
- Klaim vendor masuk akal secara prinsip.** Pada mikrokernel, driver adalah proses user mode. Jika driver layar sentuh *crash*, mikrokernel dan layanan lain tetap hidup, dan driver dapat dijalankan ulang. *Trade-off*-nya adalah overhead komunikasi antarproses. Untuk kiosk dengan beban ringan, overhead ini umumnya tidak menjadi masalah. Tetap minta bukti: uji coba, dokumentasi, dan referensi pelanggan lain.
- Rekomendasi:** hanya memasang driver dari repositori resmi distribusi atau vendor perangkat keras; menguji pembaruan di server *staging* sebelum ke produksi; mengaktifkan Secure Boot

agar modul kernel yang tidak ditandatangani ditolak; dan menyiapkan prosedur *rollback*, misalnya memilih kernel lama dari menu GRUB.

G. RANGKUMAN

- OS menyediakan layanan bagi **pengguna** (antarmuka, eksekusi program, I/O, berkas, komunikasi, deteksi kesalahan) dan bagi **efisiensi sistem** (alokasi sumber daya, pencatatan, proteksi).
- **System call** adalah pintu resmi program ke kernel. Program biasanya memanggil **API** (glibc, Win32) yang di dalamnya memanggil system call.
- Alur system call: nomor layanan + argumen → trap ke kernel mode → tabel system call → validasi → kerjakan → kembali ke user mode.
- System call dikelompokkan menjadi kontrol proses, manajemen berkas, manajemen perangkat, pemeliharaan informasi, komunikasi, dan proteksi.
- System call punya biaya. **Buffering** mengurangi jumlahnya, tetapi ada *trade-off* dengan ketahanan data.
- Struktur OS: **monolitik** (cepat, isolasi rendah), **berlapis** (rapi, lambat), **mikrokernel** (andal, overhead IPC), **modular** (fleksibel), dan **hibrida** (gabungan). Linux monolitik-modular; Windows dan macOS hibrida.
- **Boot**: firmware (UEFI/BIOS) → POST → bootloader (GRUB) → kernel + initramfs → mount root → PID 1 (systemd) → layanan siap.

H. GLOSARIUM

Istilah	Arti Singkat
API	<i>Application Programming Interface</i> , kumpulan fungsi pustaka yang dipakai pemrogram, misalnya POSIX dan Win32
POSIX	Standar antarmuka UNIX sehingga program dapat berpindah antar-OS mirip UNIX
glibc	Pustaka C standar pada kebanyakan distribusi Linux
Tabel system call	Tabel di kernel yang memetakan nomor system call ke rutin yang menanganinya
File descriptor	Bilangan bulat yang mewakili berkas atau koneksi yang sedang dibuka oleh proses
Shell	Program user mode yang membaca perintah teks dan memintanya dijalankan kernel
Monolitik	Struktur kernel di mana semua layanan berada dalam satu program di kernel mode
Mikrokernel	Struktur kernel minimal; layanan lain berjalan di user mode dan berkomunikasi lewat pesan
IPC	<i>Inter-Process Communication</i> , mekanisme pertukaran data antarproses
Modul kernel	Bagian kernel yang dapat dimuat atau dilepas saat sistem berjalan
Hibrida	Struktur yang menggabungkan konsep monolitik dan mikrokernel
Firmware	Perangkat lunak di chip motherboard yang pertama kali berjalan saat komputer menyala (BIOS/UEFI)

Istilah	Arti Singkat
Bootloader	Program yang memuat kernel ke memori, misalnya GRUB
initramfs	Sistem berkas sementara di RAM yang membantu kernel menemukan disk utama saat boot
systemd	Proses pertama (PID 1) di Ubuntu yang menyalakan dan mengelola layanan
Secure Boot	Fitur UEFI yang hanya menjalankan bootloader dan kernel yang ditandatangani secara digital
strace	Alat Linux untuk menampilkan system call yang dipanggil sebuah program

J. LATIHAN MANDIRI

1. Jelaskan perbedaan API dan system call. Berikan satu contoh fungsi API yang memanggil system call dan satu yang tidak.
2. Uraikan langkah-langkah yang terjadi sejak program memanggil `read()` hingga data diterima program. Sebutkan peran tabel system call dalam alur tersebut.
3. Mengapa kernel harus memvalidasi alamat memori yang dikirim program sebagai argumen system call? Apa akibatnya jika tidak?
4. Bandingkan struktur monolitik dan mikrokernel dari sisi kinerja dan keandalan. Dalam situasi apa Anda akan memilih masing-masing?
5. Jelaskan peran initramfs dalam proses boot. Mengapa kernel tidak langsung memasang disk utama saja?

K. PENUGASAN TERSTRUKTUR: LATIHAN KASUS 1 — ANALISIS PERBANDINGAN STRUKTUR OS

Bentuk: individu. **Batas pengumpulan:** sebelum Pertemuan 3, melalui LMS (PDF, maksimal 3 halaman di luar lampiran). **Bobot:** bagian dari komponen Latihan Kasus & Laporan Praktikum.

Skenario: Sebuah rumah sakit daerah (fiktif) akan membangun tiga sistem baru: (1) server rekam medis elektronik yang diakses ratusan tenaga kesehatan, (2) perangkat pemantau pasien di ruang ICU yang harus merespons dalam hitungan milidetik, dan (3) kiosk pendaftaran mandiri di lobi. Anda diminta memberi rekomendasi sebagai analis SI.

Tugas:

1. Untuk setiap sistem, identifikasi kebutuhan utamanya: kinerja, keandalan/isolasi kesalahan, keamanan, kemudahan pemeliharaan, atau ketersediaan SDM.
2. Rekomendasikan **struktur OS** (monolitik-modular, mikrokernel, atau hibrida) dan **satu contoh OS nyata** untuk setiap sistem. Gunakan sumber resmi (dokumentasi vendor atau jurnal) minimal satu untuk tiap rekomendasi.
3. Jelaskan alasan setiap rekomendasi dengan mengaitkannya pada materi Bagian D, termasuk *trade-off* yang harus diterima.
4. Sajikan ringkasan dalam tabel perbandingan: sistem, kebutuhan utama, struktur OS, contoh OS, alasan, risiko.
5. Cantumkan daftar pustaka. Jika memakai AI generatif, sebutkan alat dan bagian yang dibantu.

Aspek Penilaian	Bobot
Ketepatan identifikasi kebutuhan tiap sistem	20%
Ketepatan rekomendasi struktur dan contoh OS	25%
Kualitas argumen dan analisis trade-off (dikaitkan dengan teori)	35%
Penggunaan sumber yang kredibel dan sitasi	10%
Kerapian penyajian (tabel, bahasa)	10%

L. PANDUAN PRAKTIKUM 2: SSH, HIERARKI DIREKTORI, PENGGUNA, PAKET, DAN strace

L.1 Tujuan

1. Mengakses server dari komputer host melalui SSH, seperti cara administrator mengelola server sungguhan.
2. Mengenali fungsi direktori utama pada hierarki berkas Linux.
3. Membuat dan mengelola akun pengguna serta memasang dan menghapus paket dengan **apt**.
4. Mengamati system call secara langsung dengan **strace**, dan mengukur dampak buffering terhadap jumlah system call.
5. Mengamati jejak struktur kernel (modul) dan proses boot pada sistem nyata.

L.2 Langkah 1 — Mengakses VM Melalui SSH

Adapter NAT membuat VM tidak dapat dihubungi langsung dari host. Kita buat **port forwarding**: koneksi ke port 2222 di host diteruskan ke port 22 (SSH) di VM.

1. Matikan VM (**sudo poweroff**). Di VirtualBox, buka **Settings** → **Network** → **Adapter 1 (NAT)** → **Advanced** → **Port Forwarding**.
2. Tambahkan aturan: Name **ssh**, Protocol TCP, Host IP **127.0.0.1**, Host Port **2222**, Guest IP dikosongkan, Guest Port **22**.
3. Nyalakan VM. Pastikan layanan SSH aktif dengan perintah di bawah ini.
4. Dari komputer host, buka terminal (Windows 10/11: PowerShell sudah memiliki klien OpenSSH; macOS/Linux: Terminal), lalu hubungkan.

```
# di VM
systemctl status ssh      # harus "active (running)"; tekan q untuk keluar

# di komputer host
ssh -p 2222 namaNIM@127.0.0.1
# ketik "yes" saat pertama kali ditanya fingerprint, lalu masukkan password
```

Mulai sekarang, kerjakan praktikum dari jendela SSH ini. Anda dapat menyalin dan menempel perintah dengan mudah.

L.3 Langkah 2 — Hierarki Direktori Linux

Linux mengikuti *Filesystem Hierarchy Standard* (FHS). Semua berkas berada di bawah satu akar, yaitu **/**. Tidak ada drive C: atau D: seperti di Windows.

Direktori	Isi	Contoh
/	Akar seluruh sistem berkas	—
/usr/bin	Program yang dapat dijalankan pengguna (/bin di Ubuntu modern hanya tautan ke sini)	ls, cp, python3
/usr/sbin	Program administrasi sistem	sshd, useradd
/etc	Berkas konfigurasi sistem	/etc/passwd, /etc/ssh/sshd_config
/home	Direktori pribadi pengguna biasa	/home/namaNIM
/root	Direktori pribadi pengguna root	—
/var	Data yang terus berubah: log, antrian, cache, data basis data	/var/log/syslog, /var/lib/postgresql
/tmp	Berkas sementara, dapat terhapus saat reboot	—
/boot	Kernel dan initramfs untuk proses boot	vmlinuz-*, initrd.img-*
/dev	Berkas yang mewakili perangkat	/dev/sda, /dev/null
/proc, /sys	Sistem berkas virtual berisi informasi kernel dan proses (tidak ada di disk)	/proc/cpuinfo, /proc/1/status
/opt, /srv	Aplikasi tambahan dari pihak ketiga dan data layanan	—

```
ls /
ls -l /bin          # perhatikan tanda panah: /bin -> usr/bin
ls /boot            # kernel & initramfs
cat /proc/version   # informasi kernel dari sistem berkas virtual
df -h /             # kapasitas disk root
```

L.4 Langkah 3 — Manajemen Pengguna dan Grup

```
sudo adduser staf1      # buat pengguna baru (isi password; data lain boleh
                        # dikosongkan)
grep staf1 /etc/passwd # data akun (tanpa password)
sudo grep staf1 /etc/shadow # hash password (hanya root)
id staf1               # UID, GID, dan grup
sudo usermod -aG sudo staf1 # tambahkan ke grup sudo (hak administrator)
id staf1               # bandingkan dengan sebelumnya
su - staf1             # coba masuk sebagai staf1, lalu ketik: exit
sudo deluser --remove-home staf1 # hapus pengguna beserta direktori home-nya
```

Catatan Keamanan

Opsi **-a** pada **usermod -aG** artinya *append* (tambahkan). Jika lupa **-a**, pengguna akan **dikeluarkan dari semua grup lain**. Kesalahan kecil seperti ini sering terjadi di server produksi. Berikan hak **sudo** hanya kepada pengguna yang benar-benar membutuhkannya (prinsip *least privilege*, Pertemuan 13).

L.5 Langkah 4 — Manajemen Paket dengan apt

```
sudo apt update          # perbarui daftar paket dari repositori
apt search htop          # cari paket
apt show htop            # informasi paket
sudo apt install -y htop strace # pasang paket (strace dipakai di langkah berikut)
dpkg -L htop | head      # berkas apa saja yang dipasang paket itu
apt list --installed | wc -l # jumlah paket terpasang
sudo apt remove -y htop  # hapus paket (konfigurasi disimpan)
sudo apt purge -y htop   # hapus paket beserta konfigurasinya
```

Pengelola paket memastikan perangkat lunak berasal dari **repositori resmi yang ditandatangani** dan otomatis memasang ketergantungannya. Ini jauh lebih aman daripada mengunduh program dari situs sembarangan (ingat studi kasus driver pada Bagian F).

L.6 Langkah 5 — Mengamati System Call dengan strace

```
strace -c ls /etc > /dev/null          # ringkasan: system call apa saja & berapa kali
strace -e trace=openat,read,write,close cat /etc/hostname
strace -e trace=execve ls              # system call pertama yang memuat program
strace -c -f bash -c "echo halo > /tmp/x.txt"
```

Pada keluaran perintah kedua, perhatikan baris `openat(...) = 3`. Angka 3 adalah **file descriptor** yang dikembalikan kernel. Angka 0, 1, dan 2 sudah dipakai untuk *stdin*, *stdout*, dan *stderr*. Perhatikan juga bahwa `read()` dan `write()` memakai angka 3 dan 1 tersebut.

L.7 Langkah 6 — Eksperimen: Dampak Buffering pada Jumlah System Call

Buat dua program Python yang sama-sama menulis 10.000 baris log. Program pertama memanggil `os.write()` untuk setiap baris, sedangkan program kedua memakai berkas Python biasa yang otomatis memakai buffer.

```
nano tanpa_buffer.py
# --- isi berkas ---
import os
fd = os.open("log_a.txt", os.O_WRONLY | os.O_CREAT | os.O_TRUNC)
for i in range(10000):
    os.write(fd, f"transaksi {i} berhasil\n".encode())
os.close(fd)
```

```
nano dengan_buffer.py
# --- isi berkas ---
with open("log_b.txt", "w") as f:
    for i in range(10000):
        f.write(f"transaksi {i} berhasil\n")
```

```
strace -c -e trace=write python3 tanpa_buffer.py
strace -c -e trace=write python3 dengan_buffer.py
time python3 tanpa_buffer.py
time python3 dengan_buffer.py
wc -l log_a.txt log_b.txt      # pastikan isi keduanya sama-sama 10.000 baris
```

Catat jumlah panggilan `write` (kolom *calls*) dan waktu *real* dari kedua program ke dalam tabel laporan. Jalankan `time` masing-masing tiga kali lalu ambil rata-ratanya, karena hasil satu kali pengukuran dapat dipengaruhi aktivitas lain di komputer.

L.8 Langkah 7 — Jejak Struktur Kernel dan Proses Boot

```
lsmod | head -15          # modul kernel yang sedang dimuat
lsmod | wc -l             # jumlah modul
modinfo ext4 | head -5    # informasi satu modul (sistem berkas ext4)
systemctl get-default     # target boot default
systemd-analyze           # total waktu boot: firmware, loader, kernel, userspace
systemd-analyze blame | head # layanan yang paling lama saat boot
systemctl is-enabled ssh  # apakah SSH menyala otomatis saat boot?
journalctl -b | head -30  # log sejak boot terakhir
```

Catatan: pada VM, bagian *firmware* dan *loader* di keluaran **systemd-analyze** mungkin tidak muncul, tergantung firmware virtual yang dipakai. Hal ini normal.

L.9 Tugas dan Format Laporan Praktikum 2

Setiap tangkapan layar harus memperlihatkan *prompt* yang memuat *username*/NIM Anda. Jawablah pertanyaan berikut:

1. Tunjukkan bukti Anda berhasil login melalui SSH dari host. Mengapa administrator lebih memilih SSH daripada bekerja langsung di layar server?
2. Sebutkan lima system call yang paling sering dipanggil oleh **ls /etc** menurut **strace -c**. Kelompokkan masing-masing ke dalam jenis system call pada Tabel Bagian C.6.
3. Dari eksperimen buffering, sajikan tabel jumlah panggilan **write** dan rata-rata waktu kedua program. Jelaskan hasilnya dengan konsep biaya system call. Sebutkan satu risiko buffering bagi aplikasi transaksi.
4. Berapa jumlah modul kernel yang dimuat di VM Anda? Apakah keberadaan modul berarti Linux adalah mikrokernel? Jelaskan.
5. Berapa total waktu boot VM Anda, dan tiga layanan apa yang paling lama? Mengapa informasi ini penting bagi pengelola server SI?

Susunan laporan (PDF): sampul; tujuan; langkah kerja beserta tangkapan layar; jawaban pertanyaan; kesimpulan. Dikumpulkan melalui LMS sebelum Pertemuan 3.

L.10 Pemecahan Masalah Umum

Masalah	Solusi
Connection refused saat SSH	Pastikan VM menyala, layanan ssh aktif, dan aturan port forwarding benar (host 2222 → guest 22).
Permission denied saat SSH	Periksa <i>username</i> dan password. Username adalah akun di VM, bukan akun komputer host.
Peringatan REMOTE HOST IDENTIFICATION HAS CHANGED	Terjadi jika VM dipasang ulang. Hapus kunci lama dengan ssh-keygen -R "[127.0.0.1]:2222" , lalu hubungkan kembali.
strace: command not found	Pasang dengan sudo apt install -y strace .
Keluaran strace terlalu panjang	Gunakan -c untuk ringkasan, -e trace=... untuk memilih system call tertentu, atau -o hasil.txt untuk menyimpan ke berkas.
Port 2222 sudah dipakai di host	Gunakan port lain, misalnya 2223, dan sesuaikan perintah ssh -p .

M. DAFTAR PUSTAKA

- [1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.
- [2] Tanenbaum, A. S., & Bos, H. (2023). *Modern operating systems* (5th ed.). Pearson.
- [3] Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.
- [4] Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books. <https://pages.cs.wisc.edu/~remzi/OSTEP/>
- [5] Nemeth, E., Snyder, G., Hein, T. R., Whaley, B., & Mackin, D. (2017). *UNIX and Linux system administration handbook* (5th ed.). Addison-Wesley.
- [6] Kerrisk, M. (n.d.). *syscalls(2) — Linux manual page*. <https://man7.org/linux/man-pages/man2/syscalls.2.html>
- [7] Canonical Ltd. (2026). *Ubuntu 26.04 LTS release notes*. <https://documentation.ubuntu.com/release-notes/26.04/>