SISTEM OPERASI KOMPUTER ERA MODERN: KONSEP, DESAIN, DAN IMPLEMENTASI

Teguh Wahyono
Lucky Indra Kesuma
Lutfi Syafirullah
Herman Heriadi
Purwita Sari

GET PRESS INDONESIA

SISTEM OPERASI KOMPUTER ERA MODERN: KONSEP, DESAIN, DAN IMPLEMENTASI

Penulis :
Teguh Wahyono
Lucky Indra Kesuma
Lutfi Syafirullah
Herman Heriadi
Purwita Sari

ISBN :

Editor : Diana Purnama Sari., SE.M.E
Penyunting : Tri Putri Wahyuni., S.Pd
Desain Sampul dan Tata Letak : Atyka Trianisa, S.Pd

Penerbit : GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022

Redaksi :
Jln. Palarik Air Pacah No 26 Kel. Air Pacah
Kec. Koto Tangah Kota Padang Sumatera Barat
Website : www. getpress.co.id
Email : adm.getpress@gmail.com

Cetakan pertama, Juli 2024

KATA PENGANTAR

Segala Puji dan syukur atas kehadirat Allah SWT dalam segala kesempatan. Sholawat beriring salam dan doa kita sampaikan kepada Nabi Muhammad SAW. Alhamdulillah atas Rahmat dan Karunia-Nya penulis telah menyelesaikan Buku Sistem Operasi Komputer Era Modern: Konsep, Desain, Dan Implementasi ini.

Buku Ini Membahas Pengantar Sistem Operasi, Desain Dan Arsitektur Sistem Operasi, Manajemen Proses Dan Thread, Manajemen Memori Dan Penyimpanan, Keamanan Dan Kinerja.

Proses penulisan buku ini berhasil diselesaikan atas kerjasama tim penulis. Demi kualitas yang lebih baik dan kepuasan para pembaca, saran dan masukan yang membangun dari pembaca sangat kami harapkan.

Penulis ucapkan terima kasih kepada semua pihak yang telah mendukung dalam penyelesaian buku ini. Terutama pihak yang telah membantu terbitnya buku ini dan telah mempercayakan mendorong, dan menginisiasi terbitnya buku ini. Semoga buku ini dapat bermanfaat bagi masyarakat Indonesia.

Padang, Juli 2024

Penulis

DAFTAR ISI

KATA PENGANTAR ..... **i**
DAFTAR ISI ..... **ii**
DAFTAR TABEL ..... **v**
DAFTAR GAMBAR ..... **vi**
BAB 1 PENGANTAR SISTEM OPERASI MODERN **1**
1.1 Pengertian Sistem Operasi 1
1.2 Peran Penting Sistem Operasi 3
1.2.1 Efisiensi Operasional Perangkat 3
1.2.2 Stabilitas dan Reliabilitas Sistem 3
1.2.3 Keamanan Data 4
1.2.4 Kemudahan Penggunaan 5
1.2.5 Kompatibilitas Perangkat 6
1.3 Fungsi Utama Sistem Operasi 7
1.3.1 Manajemen Sumber Daya 8
1.3.2 Manajemen Proses 8
1.3.3 Manajemen Memori 9
1.3.4 Manajemen Penyimpanan 10
1.3.5 Manajemen Input/Output 11
1.4 Klasifikasi Sistem Operasi 12
1.4.1 Klasifikasi Berdasarkan Jenis Perangkat 12
1.4.2 Klasifikasi Berdasarkan Antarmuka Pengguna 13
1.4.3 Klasifikasi Berdasarkan Arsitektur 15
1.4.4 Klasifikasi Berdasarkan Jumlah Pengguna dan Proses. 15
1.4.5 Berdasarkan *Source Code*-nya 16
DAFTAR PUSTAKA 18
BAB 2 DESAIN DAN ARSITEKTUR SISTEM OPERASI **19**
2.1 Struktur Umum Sistem Operasi 19
2.2 Tinjauan Tentang Arsitektur Monolitik, Mikrokernel, dan Hibrida 25
2.3 Konsep dan Prinsip Desain yang Mendasari Pengembangan Sistem Operasi Modern 28
DAFTAR PUSTAKA 31
BAB 3 MANAJEMEN PROSES DAN THREAD **33**
3.1 Pendahuluan 33

3.2 Manajemen Proses 34
3.2.1 Process Control Block 36
3.2.2 Status Proses 48
3.2.3 Pembuatan Proses 49
3.2.4 Terminasi Proses 51
3.2.5 Komunikasi dan Hubungan Antar Proses 53
3.2.6 Sinkronisasi 56
3.2.7 Penundaan 57
3.2.8 Buffering 58
3.2.9 Manajemen Proses Sistem Operasi Windows 60
3.3 Thread 60
3.3.1 *Thread Concept* 60
3.3.2 Multithreading Models 65
3.3.3 Threading Issues 67
DAFTAR PUSTAKA 71
BAB 4 MANAJEMEN MEMORI DAN PENYIMPANAN 73
4.1 Pendahuluan 73
4.2 Prinsip Manajemen Memori Virtual dan Fisik 74
4.2.1 Memori Virtual 74
4.2.2 Memori Fisik 76
4.2.3 Perbedaan Memori Virtual dan Fisik 77
4.2.4 Prinsip – Prinsip Manajemen Memori 78
4.3 Strategi Alokasi Memori dan Manajemen *Swapping* 80
4.3.1 Alokasi Memori 80
4.3.2 Manajemen *Swapping* 82
4.4 Manajemen Penyimpanan 85
4.4.1 File system 86
4.4.2 *Disk Management* 88
DAFTAR PUSTAKA 92
BAB 5 KEAMANAN DAN KINERJA 93
5.1 Konsep Dasar Keamanan Sistem Operasi 93
5.1.1 CIA Triad (*Confidentiality, Integrity, Availability*) . 93
5.1.2 Prinsip-prinsip Keamanan Dasar 94
5.1.3 Model Keamanan Sistem Operasi 101
5.1.4 Ancaman dan Serangan pada Sistem Operasi 103
5.2 Kinerja Sistem Operasi: Konsep dan Metrik 104
5.2.1 Definisi Kinerja Sistem Operasi 104

5.2.2 Metrik Pengukuran Kinerja 105
5.3 Optimisasi Kinerja Sistem Operasi 106
5.3.1 Manajemen proses dan penjadwalan 106
5.3.2 Optimasi penggunaan memori 107
5.3.3 Teknik caching dan buffering 108
5.4 Alat-Alat yang Digunakan Untuk Analisis Kinerja Pada Keamanan Sistem Operasi 108
DAFTAR PUSTAKA 111
BIODATA PENULIS

DAFTAR TABEL

Tabel 1.1. Perbedaan antara GUI dan CLI 14
Tabel 4.1. Perbedaan Memori Virtual dan Fisik 77

DAFTAR GAMBAR

Gambar 1.1. Sistem Operasi: diantara perangkat keras dan perangkat lunak 2
Gambar 1.2. Berbagai jenis Sistem Operasi 12
Gambar 1.3. GUI (kiri) vs CLI (kanan) 13
Gambar 1.4. *Single tasking operating system* *15*
Gambar 1.5. *Multi tasking operating system* *16*
Gambar 3.1. Process Control Block (PCB) 36
Gambar 3.2. Diagram Status Proses 48
Gambar 3.3. Model Single Thread dan Multi Thread 61
Gambar 3.4. Model Many-to-One (M:1) 65
Gambar 3.5. Model One-to-One (1:1) 66
Gambar 3.6. Model Many-to-Many (M:M) 67
Gambar 4.1. Cara Kerja Memori Virtual 75
Gambar 4.2. Memori Fisik 76
Gambar 4.3. Proses *Swapping* *83*

BAB 1
PENGANTAR SISTEM OPERASI MODERN

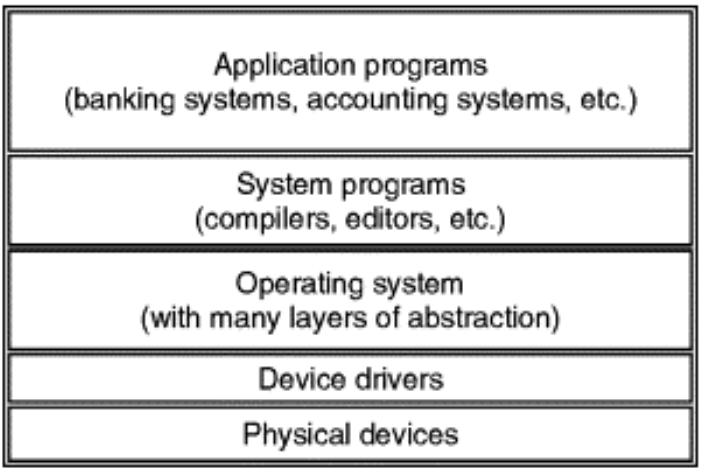
Oleh Teguh Wahyono

1.1 Pengertian Sistem Operasi

Sistem operasi merupakan perangkat lunak yang sangat penting dalam dunia komputasi modern. Tanpa adanya sistem operasi, komputer dan perangkat teknologi tidak akan dapat berfungsi dengan baik. Sistem operasi menjalankan dan mengelola perangkat keras komputer serta menyediakan berbagai layanan dan platform bagi aplikasi perangkat lunak lainnya. Sistem operasi modern tidak hanya berfungsi sebagai jembatan antara perangkat keras dan pengguna, tetapi juga menawarkan berbagai fitur yang mendukung keamanan, konektivitas, serta kemudahan penggunaan di lingkungan yang kompleks dan terdistribusi.

Beberapa ahli mendefinisikan pengertian sistem operasi secara berbeda, walaupun memiliki makna yang sama. Lien (2005) dalam buku *Modern Operating System* mendefinisikan sistem operasi sebagai perangkat lunak yang mengendalikan semua sumber daya komputer dan menyediakan basis tempat program aplikasi dapat ditulis. Sedangkan Mcleod (2006) menyebutkan bahwa sistem operasi adalah program yang melakukan manajemen terhadap sumber daya perangkat keras dan sumber daya perangkat lunak komputer. Abdul Kadir (2013) mengatakan bahwa sistem operasi merupakan program terpenting dari program-program lain yang terdapat dalam sistem komputer. Sementara itu Purbo (2019) menyebut sistem operasi sebagai suatu sistem yang terdiri atas beberapa komponen perangkat lunak atau software yang memiliki fungsi untuk mengontrol seluruh aktivitas yang dilakukan komputer.

Perbedaan antara sistem operasi dengan perangkat lain dalam sistem komputer nampak pada hirarki yang ditujukkan pada Gambar 1.1 (Lien, 2005).



Gambar 1.1. Sistem Operasi: diantara perangkat keras dan perangkat lunak

Gambar 1.1 menunjukkan sistem operasi merupakan jembatan penghubung antara perangkat keras dan perangkat lunak aplikasi. Pada hirarki yang paling atas adalah program aplikasi, yaitu perangkat lunak yang digunakan user untuk membantu pekerjaan sehari-hari, seperti aplikasi perkantoran, aplikasi perbankan, sistem informasi dan sebagainya. Sistem operasi bersama dengan *system program* dan *device driver* bekerja untuk mengoperasionalisasikan perangkat sehingga aplikasi dapat digunakan untuk bekerja.

Sistem operasi di era komputer modern memainkan peran yang sangat penting dalam mendukung kinerja, keamanan, dan kemudahan penggunaan perangkat komputasi. Dengan manajemen sumber daya yang canggih, fitur keamanan yang kuat, antarmuka pengguna yang intuitif, dan dukungan konektivitas yang luas, OS modern memungkinkan pengguna untuk memanfaatkan teknologi dengan lebih efektif dan efisien. Seiring dengan perkembangan teknologi, sistem operasi akan terus beradaptasi dan berkembang

untuk memenuhi kebutuhan yang semakin kompleks dari pengguna dan lingkungan komputasi yang terus berubah.

1.2 Peran Penting Sistem Operasi

Sistem operasi memiliki peranan penting untuk melakukan efisiensi operasional perangkat, stabilitas dan keandalan sistem, keamanan data, antar muka pengguna dengan perangkat, serta jembatan kompatibilitas antar perangkat yang terhubung (Silberschatz et.al, 2021). Berikut adalah penjelasan dari beberapa tersebut.

1.2.1 Efisiensi Operasional Perangkat

Sistem operasi (OS) adalah perangkat lunak penting yang mengelola perangkat keras dan perangkat lunak pada komputer, sehingga perannya sangat krusial untuk memastikan efisiensi operasional perangkat. Sistem operasi memungkinkan komputer menjalankan berbagai aplikasi secara efisien. Ini membantu dalam meningkatkan produktivitas pengguna dengan memungkinkan *multitasking* dan penggunaan sumber daya secara optimal.

Sistem operasi mengelola berbagai proses yang berjalan pada komputer, seperti penjadwalan tugas, alokasi waktu CPU, dan pengelolaan *multitasking*. Dengan mengatur proses secara efisien, sistem operasi memastikan bahwa tidak ada konflik antara tugas yang berbeda dan setiap proses mendapatkan sumber daya yang dibutuhkannya.

Disamping itu sistem operasi juga mengatur penggunaan memori komputer, baik *Random Access Memory* (RAM) maupun *virtual memory*. Hal tersebut melibatkan alokasi memori untuk program yang sedang berjalan dan memastikan bahwa tidak ada program yang menggunakan memori yang dialokasikan untuk program lain. Manajemen memori yang baik mencegah penggunaan memori yang berlebihan dan memastikan kinerja yang optimal.

1.2.2 Stabilitas dan Reliabilitas Sistem

Sistem operasi yang baik memastikan stabilitas dan keandalan sistem, serta mengurangi kemungkinan *crash* atau kegagalan sistem yang dapat mengganggu pekerjaan pengguna.

Sistem operasi melakukan manajemen sumber daya yang efisien. isolasi dan proteksi, manajemen pembaruan sistem, manajemen jaringan dan pengelolaan sistem file yang memastikan bahwa komputer dapat beroperasi dengan stabil dan andal, bahkan di bawah kondisi beban kerja yang berat atau saat menghadapi permasalahan sistem.

Sistem operasi secara berkala menerima pembaruan untuk memperbaiki *bug*, dan meningkatkan kinerja. Hal ini membantu menjaga sistem tetap stabil dan andal dengan menutup celah keamanan dan memperbaiki masalah yang ada. Sistem operasi modern juga memiliki fitur pemulihan yang memungkinkan pengguna untuk mengembalikan sistem ke keadaan sebelumnya jika terjadi masalah serius setelah pembaruan atau perubahan sistem.

Sistem operasi juga mengelola koneksi jaringan untuk memastikan komunikasi yang stabil dan andal antara perangkat. Hal ini mencakup pengaturan protokol jaringan, manajemen lalu lintas, dan pengendalian akses. Sistem operasi dapat mendeteksi masalah jaringan dan mencoba memperbaikinya atau mengalihkan lalu lintas untuk menjaga konektivitas.

1.2.3 Keamanan Data

Sistem operasi modern dilengkapi dengan fitur keamanan yang kuat untuk melindungi data pengguna dari ancaman seperti virus, malware, dan akses yang tidak sah. Sistem operasi melakukan kontrol terhadap akses sistem, seperti otentikasi pengguna. Sistem operasi memastikan bahwa hanya pengguna yang terverifikasi yang dapat mengakses sistem. Ini dilakukan melalui mekanisme otentikasi seperti kata sandi, biometrik, dan kartu pintar. Sistem iperasi juga mengelola izin akses untuk file, folder, serta sumber daya sistem lainnya. Pengguna diberikan hak akses tertentu (*read, write*, *execute*) berdasarkan kebutuhan serta mencegah akses tidak sah ke data sensitif.

Dalam hal keamanan jaringan, sistem operasi sering kali dilengkapi dengan *firewall* yang dapat digunakan untuk mengontrol lalu lintas jaringan masuk dan keluar, memblokir akses yang mencurigakan atau tidak diinginkan. Sistem operasi juga

mendukung penggunaan *Virtual Private Network* (VPN) untuk memastikan bahwa data yang dikirim melalui jaringan publik dienkripsi dan aman dari penyadapan.

Sistem operasi juga sering kali menyertakan atau mendukung perangkat lunak antivirus dan antimalware yang memindai, mendeteksi, dan menghapus perangkat lunak berbahaya dari sistem. Sistem operasi dapat menjalankan aplikasi dalam lingkungan terisolasi (*sandbox*) untuk mencegah aplikasi berbahaya mempengaruhi sistem utama.

Sistem operasi mencatat log aktivitas pengguna dan sistem, termasuk upaya akses yang berhasil dan gagal, perubahan pada file penting, dan aktivitas jaringan. Log ini penting untuk mendeteksi dan menganalisis pelanggaran keamanan. Sistem operasi juga memungkinkan audit keamanan yang memeriksa konfigurasi dan kebijakan keamanan untuk memastikan kepatuhan terhadap standar keamanan yang ditetapkan.

1.2.4 Kemudahan Penggunaan

Sistem Operasi dirancang dengan antarmuka pengguna yang intuitif, sehingga membuat penggunaan komputer menjadi lebih mudah dan menyenangkan, bahkan bagi yang tidak memiliki latar belakang teknis. Sistem operasi modern menyediakan *Graphical User Interface* (GUI) yang memungkinkan pengguna berinteraksi dengan komputer melalui ikon, jendela, dan menu yang mudah dipahami, menggantikan perintah teks yang kompleks. Pengguna bahkan bisa melakukan personalisasi, dimana pengguna dapat menyesuaikan tampilan dan perilaku antarmuka sesuai dengan preferensi masing-masing, termasuk misalnya mengubah *wallpaper*, tema, tata letak, dan warna.

Terkait kebutuhan instalasi dan penghapusan aplikasi, sistem operasi menyediakan mekanisme yang mudah untuk hal tersebut. Beberapa sistem operasi modern menyediakan *application store* terintegrasi yang memudahkan pengguna untuk menemukan dan mengunduh aplikasi yang diinginkannya.

Sistem operasi juga mendukung *multitasking* yang memungkinkan pengguna menjalankan beberapa aplikasi sekaligus dan beralih antar aplikasi dengan lancar melalui *taskbar* atau

switcher aplikasi. Sistem juga menyediakan file explorer yang memungkinkan pengguna menjelajahi, mengatur, dan mengelola file dan folder dengan mudah melalui antarmuka yang intuitif. Fitur pencarian yang cepat dan efisien memungkinkan pengguna menemukan file, aplikasi, dan pengaturan dengan cepat hanya dengan memasukkan beberapa kata kunci.

Untuk semakin memudahkan pengoperasiona, banyak sistem operasi modern menyertakan asisten digital berbasis AI, seperti Cortana di Windows atau Siri di macOS, yang dapat membantu pengguna dengan tugas-tugas sehari-hari melalui perintah suara. Sistem juga mendukung automatisasi tugas dengan alat-alat seperti skrip dan makro, memungkinkan pengguna mengotomatiskan rutinitas yang berulang dan meningkatkan produktivitas. Dengan berbagai fitur dan fungsi di atas, sistem operasi memastikan bahwa pengguna dapat berinteraksi dengan komputer mereka dengan cara yang mudah, efisien, dan intuitif, meningkatkan produktivitas dan pengalaman pengguna secara keseluruhan.

1.2.5 Kompatibilitas Perangkat

Sistem operasi memainkan peran penting dalam memastikan kompatibilitas perangkat, memungkinkan berbagai perangkat keras dan perangkat lunak untuk bekerja bersama secara harmonis. Sistem operasi mengelola *driver* yang merupakan program khusus yang memungkinkan sistem operasi berkomunikasi dengan perangkat keras. *Driver* ini diterapkan oleh produsen perangkat keras untuk memastikan perangkat mereka berfungsi dengan baik di berbagai sistem operasi. Sistem operasi sering kali memiliki database driver yang luas dan dapat mengunduh serta menginstal driver yang diperlukan secara otomatis saat perangkat baru terdeteksi. Pembaruan driver juga dikelola oleh sistem operasi untuk menjaga kompatibilitas dan kinerja optimal.

Sistem operasi menyediakan *Application Programming Interface* (API) yang digunakan oleh perangkat lunak untuk berinteraksi dengan perangkat keras tanpa perlu mengetahui detail teknis perangkat tersebut. Ini memastikan bahwa aplikasi dapat

berjalan pada berbagai perangkat keras yang berbeda. Sistem operasi juga menyediakan abstraksi perangkat keras. Dengan menyediakan lapisan abstraksi antara perangkat keras dan perangkat lunak, OS memungkinkan aplikasi berfungsi dengan baik tanpa harus dirancang khusus untuk setiap jenis perangkat keras.

Dalam hal kompatibilitas perangkat lunak, sistem operasi dirancang dengan kemampuan *Backward Compatibility*, yaitu kemampuan mendukung aplikasi dan perangkat keras lama, memastikan bahwa pengguna dapat terus menggunakan perangkat mereka bahkan setelah melakukan pembaruan sistem. Beberapa sistem operasi modern juga mendukung teknologi virtualisasi dan emulasi, memungkinkan pengguna menjalankan perangkat lunak dan sistem operasi lain di atas sistem utama. Hal ini membantu dalam menjalankan aplikasi yang mungkin tidak kompatibel secara langsung dengan sistem utama.

Dalam hal kemudahan koneksi perangkat, sistem operasi mendukung teknologi *plug and play*, yang memungkinkan perangkat baru dikenali dan dikonfigurasi secara otomatis tanpa memerlukan intervensi pengguna. Ini memudahkan pemasangan perangkat baru dan memastikan kompatibilitas. Beberapa sistem operasi juga mendukung *hot swapping*, yang memungkinkan perangkat dilepas dan dipasang kembali tanpa perlu mematikan sistem, memperbaiki kenyamanan dan kompatibilitas perangkat. Dengan berbagai fungsi di atas, sistem operasi memastikan bahwa berbagai perangkat keras dan perangkat lunak dapat berfungsi bersama secara efektif dan efisien, memberikan pengalaman pengguna yang baik dan kompatibilitas yang luas.

1.3 Fungsi Utama Sistem Operasi

Fungsi utama dari sistem operasi meliputi beberapa hal seperti: (1) manajemen sumber daya, (2) manajemen proses, (3) manajemen memory, dan (4) manajemen penyimpanan, termasuk (5) antar muka pengguna dan keamanan sistem (Dusseau et al, 2023). Berikut adalah penjelasan dari beberapa fungsi tersebut.

1.3.1 Manajemen Sumber Daya

Sistem operasi bertanggung jawab untuk mengelola sumber daya perangkat keras komputer, seperti CPU, memori, penyimpanan, dan perangkat input/output.

Dengan manajemen yang efisien, sistem operasi memastikan bahwa setiap aplikasi mendapatkan sumber daya yang diperlukan tanpa mengganggu aplikasi lain. Dengan pengelolaan sumber daya yang baik, sistem operasi memastikan bahwa perangkat keras dan perangkat lunak dapat bekerja bersama dengan lancar, efisien, dan aman. Ini memungkinkan pengguna dan aplikasi untuk menggunakan sistem komputer secara efektif.

1.3.2 Manajemen Proses

Fungsi sistem operasi sebagai manajemen proses melibatkan pengelolaan berbagai aspek yang terkait dengan proses yang berjalan pada komputer. Sistem operasi mengelola eksekusi proses, termasuk penjadwalan tugas, pengalokasian waktu CPU, dan penanganan interupsi. Ini memungkinkan banyak aplikasi berjalan secara bersamaan (*multitasking*) dan memastikan bahwa setiap aplikasi berfungsi dengan baik (Remzi, 2023).

Dalam hal penjadwalan proses (*process scheduling*) sistem operasi menentukan proses mana yang akan dijalankan oleh CPU pada waktu tertentu. Algoritma penjadwalan yang berbeda digunakan untuk mencapai tujuan seperti keadilan (*fairness*), efisiensi CPU, dan responsivitas sistem. Kemampuan multitasking sistem memungkinkan beberapa proses berjalan secara bersamaan dengan cara membagi waktu CPU di antara mereka. Ini memastikan bahwa sistem dapat menangani banyak tugas sekaligus.

Sistem operasi juga mengatur pembuatan dan penghentian proses (*process creation and termination*). Sistem operasi menyediakan mekanisme untuk membuat proses baru. Ini termasuk mengalokasikan sumber daya yang diperlukan dan menginisialisasi tabel proses. Sistem operasi dapat menghentikan proses yang berjalan, melepaskan sumber daya yang digunakan, dan memperbarui tabel proses. Sistem operasi juga melakukan sinkronisasi dan komunikasi antar proses yaitu untuk memastikan bahwa proses yang berjalan bersamaan tidak saling mengganggu.

Komunikasi antar proses memungkinkan proses untuk berkomunikasi dan bertukar informasi. Sistem operasi menggunakan strategi untuk mencegah *deadlock*, dimana dua atau lebih proses saling menunggu sumber daya yang dipegang oleh proses lain. Dengan mengelola semua aspek tersebut, sistem operasi memastikan bahwa proses yang berjalan pada sistem komputer dapat beroperasi secara efisien dan terkoordinasi. Ini memungkinkan sistem untuk memberikan performa optimal dan pengalaman pengguna yang responsif.

1.3.3 Manajemen Memori

Fungsi sistem operasi sebagai manajemen memori mencakup pengelolaan penggunaan memori komputer oleh berbagai aplikasi dan proses. Manajemen memori adalah kunci untuk memastikan bahwa setiap proses memiliki akses ke ruang memori yang diperlukan tanpa konflik dengan proses lain. Sistem operasi mengalokasikan memori secara dinamis untuk proses yang berjalan, memastikan bahwa memori yang dibutuhkan dialokasikan saat proses dimulai dan dibebaskan saat proses selesai. Sedangkan alokasi statis dilakukan ketika mengalokasikan memori pada waktu kompilasi, di mana ukuran dan lokasi memori ditentukan sebelumnya.

Sistem operasi juga melakukan manajemen memori virtual, yaitu melakukan *paging* dan *isolation*. Paging adalah ketika memori fisik dibagi menjadi blok-blok kecil yang disebut halaman, dan memori virtual dibagi menjadi blok-blok dengan ukuran yang sama yang disebut page frames. Proses dapat dijalankan meskipun tidak semua halaman berada di memori fisik. Sedangkan *segmentation* adalah membagi memori ke dalam segmen-segmen yang berukuran variabel sesuai dengan kebutuhan proses. Setiap segmen memiliki alamat dasar dan panjang, memungkinkan isolasi dan perlindungan yang lebih baik.

Proteksi terhadap memori dilakukan dengan isolasi proses, dimana setiap proses diisolasi sehingga tidak dapat mengakses memori yang dialokasikan untuk proses lain. Ini mencegah proses yang salah atau berbahaya dari merusak data proses lain. Sedangkan mekanisme proteksi dilakukan dengan menggunakan

tabel halaman, register batas, dan bit proteksi untuk memastikan bahwa proses hanya dapat mengakses memori yang diizinkan.

Manajemen memori bersama atau *shared memory* dilakukan sehingga memungkinkan beberapa proses untuk berbagi ruang memori yang sama, memfasilitasi komunikasi antar proses dan penghematan memori. Sistem operasi mengelola akses dan sinkronisasi memori bersama untuk mencegah konflik. Dengan fungsi-fungsi tersebut, sistem operasi memastikan bahwa memori digunakan secara efisien dan proses dapat berjalan dengan lancar tanpa konflik atau kekurangan sumber daya memori, yang tentunya juga memastikan stabilitas dan keamanan sistem, serta memberikan kinerja optimal.

1.3.4 Manajemen Penyimpanan

Fungsi sistem operasi sebagai manajemen penyimpanan melibatkan pengelolaan dan pengorganisasian data yang disimpan pada perangkat penyimpanan sekunder seperti hard drive, SSD, dan media penyimpanan eksternal. Sistem operasi melakukan *File System Management*, yang diantaranya menyediakan struktur hierarkis untuk mengorganisasi file dan direktori, memungkinkan pengguna untuk mengakses dan mengelola data dengan cara yang terstruktur. Sistem operasi mendukung berbagai format sistem file seperti NTFS, FAT32, ext4, dan HFS+. Setiap format memiliki fitur dan karakteristik yang berbeda.

Pengelolaan file baik untuk pembuatan dan penghapusan file serta operasi file dasar seperti membaca, menulis, membuka, dan menutup file. Sistem operasi menyediakan fungsi untuk membuat dan menghapus file, mengelola metadata yang terkait seperti nama file, ukuran, dan waktu modifikasi. Sistem operasi mengelola hak akses untuk setiap file dan direktori, menentukan siapa yang dapat membaca, menulis, atau mengeksekusi file tersebut.

Untuk pengelolaan disk, sistem operasi bisa melakukan fungsi partisi, format dan lain sebagainya. *Partisi disk* memungkinkan pembagian disk fisik menjadi beberapa partisi logis, yang masing-masing dapat memiliki sistem file yang berbeda. Sistem juga menyediakan fungsi untuk memformat partisi disk,

menghapus semua data dan mempersiapkan partisi untuk digunakan dengan sistem file tertentu.

Fungsi *defragmentasi* dilakukan untuk mengatur ulang data pada disk untuk mengurangi fragmentasi dan meningkatkan kecepatan akses data dengan memastikan bahwa file disimpan dalam blok-blok yang berurutan.

1.3.5 Manajemen Input/Output

Fungsi sistem operasi sebagai manajemen Input/Output mencakup pengelolaan semua operasi input dan output, yang melibatkan berbagai perangkat keras seperti keyboard, mouse, printer, disk drive, monitor, dan perangkat jaringan. Sistem operasi menggunakan *driver* untuk menyediakan antarmuka yang konsisten untuk perangkat keras. *Driver* adalah perangkat lunak khusus yang mengizinkan sistem operasi berkomunikasi dengan perangkat I/O tertentu tanpa harus mengetahui detail teknis perangkat tersebut.
Sistem operasi melakukan fungsi penjadwalan *I/O Scheduling*) dengan mengatur urutan permintaan akses disk, dalam mengoptimalkan waktu akses dan *throughput*. Sistem operasi menyimpan permintaan I/O dalam antrian dan memprosesnya berdasarkan kebijakan penjadwalan tertentu untuk meningkatkan efisiensi. Sistem operasi bisa melakukan *buffering data*, menyimpan data sementara dalam *buffer* untuk mengatasi perbedaan kecepatan antara perangkat I/O dan CPU. *Buffering* membantu meningkatkan kinerja dengan memungkinkan CPU dan perangkat I/O bekerja lebih efisien.

Sistem operasi juga mendeteksi kesalahan I/O dan mengambil tindakan korektif seperti mengulang permintaan, mengarahkan ulang operasi, atau melaporkan kesalahan kepada pengguna atau aplikasi. Sistem operasi juga menyimpan log kesalahan, menyimpan catatan kesalahan I/O untuk analisis dan pemecahan masalah. Sistem operasi menyediakan fungsi operasional untuk membaca, menulis, membuka, dan menutup file, yang melibatkan operasi I/O di tingkat disk. Dengan berbagai fungsi tersebut di atas, sistem operasi memastikan bahwa operasi I/O dilakukan dengan efisien dan andal, memungkinkan perangkat keras dan perangkat lunak bekerja bersama secara efektif. Ini

memberikan pengalaman pengguna yang mulus dan meningkatkan kinerja keseluruhan sistem.

1.4 Klasifikasi Sistem Operasi

Sistem operasi dapat diklasifikasikan berdasarkan berbagai kriteria, seperti jenis perangkat yang didukung, metode interaksi pengguna, arsitektur sistem, dan tujuan penggunaan. Secara lebih lengkap, berikut adalah beberapa klasifikasi utama dari sistem operasi.

Gambar 1.2. Berbagai jenis Sistem Operasi

1.4.1 Klasifikasi Berdasarkan Jenis Perangkat.

Berdasarkan jenis perangkatnya, sistem operasi dibagi menjadi sistem operasi desktop, sistem operasi mobile, sistem operasi server dan sistem operasi perangkat *embedded*.

1. Sistem Operasi Desktop
   Sistem operasi ini dirancang untuk desktop, yang meliputi komputer pribadi atau PC. Contoh sistem operasi desktop adalah Windows, macOS, dan Linux.
2. Sistem Operasi Mobile
   Dirancang untuk perangkat seluler seperti smartphone dan tablet. Contohnya termasuk Android dan iOS.
3. Sistem Operasi Server: Dirancang untuk menjalankan server yang menyediakan layanan jaringan. Contohnya termasuk

Windows Server, Linux (seperti Ubuntu Server, CentOS), dan UNIX.

4. Sistem Operasi Embedded: Digunakan dalam perangkat khusus seperti router, smart TV, perangkat IoT, dan lain-lain. Contohnya termasuk Embedded Linux dan VxWorks.

1.4.2 Klasifikasi Berdasarkan Antarmuka Pengguna

Berdasarkan antarmuka pengguna, sistem operasi bisa dikelompokkan menjadi sistem operasi berbasis GUI (*Graphical User Interface*) dan berbasis CLI (*Command Line Interface*).

1. Sistem Operasi Berbasis GUI.
   Sistem operasi berbasis GUI menggunakan antarmuka grafis untuk interaksi pengguna. Contoh sistem operasi berbasis GUI ini adalah Windows dan macOS.
2. Sistem Operasi Berbasis CLI
   Sistem operasi berbasis CLI menggunakan antarmuka baris perintah untuk interaksi pengguna. Contohnya adalah DOS (*Disk Operating System*), serta sistem operasi Linux (seperti distribusi *Ubuntu Server*) dan UNIX.

Gambar 1.3. GUI (kiri) vs CLI (kanan)

Gambar 1.3 menunjukkan perbedaan tampilan antara GUI dengan CLI. Terlihat bahwa GUI memiliki user interface yang berbasis grafis, sedangkan CLI memiliki perintah-perintah yang berbasis teks atau *command line*. Selanjutnya Tabel 1 menunjukkan perbedaan mendasar antara GUI dengan CLI.

Tabel 1.1. Perbedaan antara GUI dan CLI

Nomor	Karakteristik	GUI	CLI
1	User interface	Menggunakan elemen grafis seperti jendela, ikon, dan menu.	Menggunakan antarmuka berbasis teks.
2	Kemudahan penggunaan	Lebih mudah digunakan, terutama bagi pemula, karena intuitif dan visual.	Pengguna harus menghafal dan memahami perintah yang diperlukan.
3	Produktivitas	Meningkatkan produktivitas untuk kebutuhan visualisasi, seperti desain grafis dan pengelolaan file.	Efisien untuk tugas-tugas yang berulang atau kompleks, dapat menggunakan skrip untuk otomatisasi
4	Fleksibilitas	Biasanya kurang fleksibel untuk tugas-tugas kompleks atau otomatisasi.	Sangat fleksibel untuk konfigurasi sistem, manajemen server, dan pengelolaan tugas-tugas dengan presisi tinggi.
5	Konsumsi Daya	Mengandalkan perangkat keras yang lebih kuat dan konsumsi memori yang lebih tinggi.	Mengkonsumsi sumber daya sistem yang lebih sedikit dibandingkan GUI.
6	Contoh	Windows, macOS, dan distribusi Linux dengan lingkungan desktop seperti GNOME atau KDE.	Bash (di Linux dan macOS), PowerShell (di Windows), dan Command Prompt (di Windows).

1.4.3 Klasifikasi Berdasarkan Arsitektur

Berdasarkan arsitekturnya, sistem operasi bisa dikelompokkan menjadi sistem operasi monolitik, mikrokernel, dan sistem operasi hybrid (Laursen, 2021). Berikut adalah penjelasan klasifikasi sistem operasi berdasarkan arsitekturnya.

1. Sistem Operasi Monolitik
   Sistem operasi monolitik memiliki karakteristik dimana semua fungsi sistem operasi berjalan dalam satu ruang dengan alamat yang sama. Contohnya adalah Sistem operasi Unix.
2. Sistem Operasi Mikrokernel
   Sistem operasi mikrokernel memiliki karakteristik dimana hanya fungsi inti yang berjalan dalam kernel, sementara fungsi lainnya berjalan dalam ruang pengguna. Contohnya adalah Minix dan QNX.
3. Sistem Operasi Hybrid
   Sistem operasi hybrid memiliki karakteristik menggabungkan elemen dari sistem monolitik dan mikrokernel. Contoh dari sistem operasi ini adalah Windows NT dan macOS.

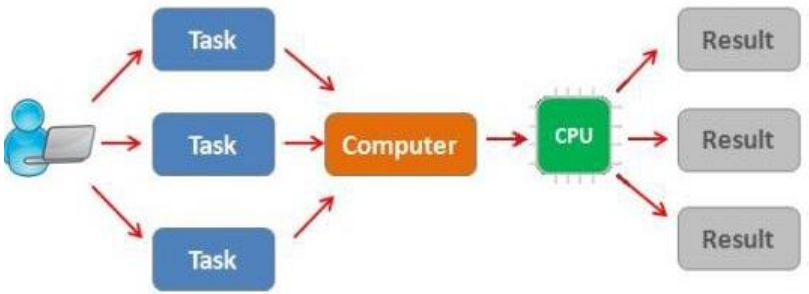
1.4.4 Klasifikasi Berdasarkan Jumlah Pengguna dan Proses.

Berdasarkan jumlah pengguna dan proses, maka sistem operasi dapat dikelompokkan menjadi sistem operasi *single user*, *multi user*, *single tasking* dan *multitasking*. Berikut adalah penjelasan lebih lanjut tentang berbagai jenis sistem operasi berdasarkan jumlah pengguna dan prosesnya.

Gambar 1.4. *Single tasking operating system*

1. Sistem Operasi Single-User
   Sistem operasi ini hanya mendukung satu pengguna pada satu waktu. Contoh dari sistem operasi *single user* adalah adalah MS-DOS.

2. Sistem Operasi Multi-User
   Sistem operasi ini mendukung adanya beberapa pengguna secara simultan. Contohnya adalah UNIX dan Windows Server.
3. Sistem Operasi *Single-Tasking*
   Sistem operasi ini hanya dapat menjalankan satu tugas pada satu waktu. Contohnya adalah MS-DOS. Gambar 1.4 menunjukkan diagram pekerjaan yang dilakukan secara single tasking.
4. Sistem Operasi *Multi-Tasking*
   Ssistem operasi ini dapat menjalankan beberapa tugas secara bersamaan. Contohnya adalah Windows dan Linux. Gambar 1.5 menunjukkan diagram pekerjaan yang dilakukan secara *multi- tasking*.



Gambar 1.5. *Multi tasking operating system*

1.4.5 Berdasarkan *Source Code*-nya

Berdasarkan *source code* atau kode sumber programnya, sistem operasi dapat dibagi menjadi distem operasi *open source* dana sistem operasi *proprietary* (Oliveira et al., 2022). Berikut adalah penjelasan sistem operasi *open source* dan *proprietary*.

1. Sistem Operasi *Open Source*
   Sistem operasi *open source* merupakan sistem operasi dengan *source code*-nya tersedia untuk umum, *free* dan dapat disebarluaskan serta dimodifikasi sesuai kebutuhan. Contoh dari sistem operasi jenis ini adalah Linux dan FreeBSD.

2. Sistem Operasi *Proprietary*
   Sistem operasi jenis ini kode sumbernya tidak tersedia untuk umum dan hanya dapat diakses atau dimodifikasi oleh pembuatnya. Sistem operasi jenis ini biasanya memiliki lisensi yang berbayar. Contoh *proprietary* adalah Windows dan macOS.

DAFTAR PUSTAKA

Abraham Silberschatz et.al, 2021, Operating System Concepts 9th Edition. Wiley Publishing.

Abdul Kadir. 2013. *Pengantar Teknologi Informasi*. Yogyakarta: Andi Offset.

de Oliveira, Daniel Bristot, Daniel Casini, and Tommaso Cucinotta. 2022. Operating system noise in the linux kernel. *IEEE Transactions on Computers* 72.1: 196-207.

Eric Laursen. 2021. *The operating system: An anarchist theory of the modern state*. AK Press.

Onno W. Purbo. 2019. *Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android*, Penerbit Andi Yogyakarta.

Raymond McLeod, George Schell. 2006. *Management Information Systems 10th Edition*, Pearson; 2006

Remzi H. Arpaci-Dusseau & Andrea C. Arpaci-Dusseau, 2023. Operating Systems: Three Easy Pieces, Arpaci-Dusseau Books Publishing.

Yao-Nan Lien, 2005. *Operating Systems, The Electrical Engineering Handbook,* Academic Press.

BAB 2
DESAIN DAN ARSITEKTUR SISTEM OPERASI

Oleh Lucky Indra Kesuma

2.1 Struktur Umum Sistem Operasi

Sistem operasi modern memiliki struktur yang kompleks namun terorganisir dengan baik. Secara umum, struktur sistem operasi terdiri dari beberapa komponen utama yang saling berinteraksi (Tanenbaum & Bos, 2015):

1. Kernel

   Kernel adalah komponen inti dari sistem operasi yang beroperasi di level paling dasar, bertindak sebagai jembatan antara aplikasi dan perangkat keras komputer (Tanenbaum & Bos, 2015). Fungsi utama kernel meliputi:

   a. Manajemen Proses
      Kernel bertanggung jawab untuk pembuatan, penjadwalan, dan penghentian proses. Ini termasuk alokasi sumber daya CPU, manajemen context switching, dan koordinasi antar proses (Silberschatz, Galvin & Gagne, 2018).
   b. Manajemen Memori
      Kernel mengelola alokasi dan dealokasi memori, termasuk implementasi virtual memory, paging, dan segmentasi (Love, 2010).
   c. Manajemen Perangkat I/O
      Kernel mengontrol akses ke perangkat I/O, mengelola driver perangkat, dan mengoptimalkan operasi I/O (Stallings, 2018).
   d. Manajemen Berkas
      Kernel mengimplementasikan sistem berkas, mengatur akses ke penyimpanan, dan mengelola metadata berkas (Tanenbaum & Bos, 2015).

2. Antarmuka Pemrograman Aplikasi (API)

   API sistem operasi menyediakan set fungsi, protokol, dan alat yang memungkinkan pengembang untuk membuat aplikasi yang dapat berinteraksi dengan sistem operasi (Silberschatz, Galvin & Gagne, 2018). Komponen utama API meliputi:

   a. System Calls
      Mekanisme yang memungkinkan program user-level meminta layanan kernel. Ini termasuk operasi seperti membuat proses baru, membaca/menulis berkas, atau mengalokasikan memori (Love, 2010).
   b. Library Functions
      Fungsi-fungsi tingkat tinggi yang membungkus satu atau lebih system calls, menyederhanakan tugas-tugas pemrograman umum (Stallings, 2018).
   c. POSIX API
      Standar API yang didefinisikan oleh IEEE untuk menjaga kompatibilitas antar sistem UNIX-like (Stevens & Rago, 2013).
   d. Win32 API
      API utama untuk sistem operasi Windows, menyediakan akses ke fungsi-fungsi sistem Windows (Russinovich, Solomon & Ionescu, 2012).

   API memainkan peran krusial dalam abstraksi perangkat keras, memungkinkan pengembang untuk menulis aplikasi yang portabel antar berbagai platform perangkat keras tanpa perlu khawatir tentang detail implementasi low-level (Tanenbaum & Bos, 2015).

3. Shell

   Shell adalah antarmuka pengguna yang memungkinkan interaksi antara pengguna dan sistem operasi. Ini bertindak sebagai penerjemah antara pengguna dan kernel sistem operasi (Robbins, 2016). Fungsi utama shell meliputi:

a. Interpretasi Perintah:
   Shell menerima perintah dari pengguna, menginterpretasikannya, dan menjalankan program yang sesuai (Shotts, 2019).
b. Scripting:
   Shell menyediakan bahasa scripting yang memungkinkan pengguna untuk mengotomatisasi tugas-tugas kompleks (Blum & Bresnahan, 2015).
c. Manajemen Proses:
   Shell memungkinkan pengguna untuk menjalankan, menghentikan, dan mengontrol proses (Robbins, 2016).
d. Redirection dan Piping:
   Shell memungkinkan pengalihan input/output dan menghubungkan output satu program ke input program lain (Shotts, 2019).

Terdapat dua jenis utama shell:

a. *Command-Line Interface* (CLI): Seperti Bash di Unix/Linux atau PowerShell di Windows, yang menerima perintah dalam bentuk teks (Blum & Bresnahan, 2015).
b. *Graphical User Interface* (GUI): Seperti GNOME di Linux atau Windows Explorer di Windows, yang menyediakan antarmuka grafis (Tanenbaum & Bos, 2015).

4. Sistem Berkas

   Sistem berkas adalah komponen sistem operasi yang mengatur bagaimana data disimpan, diakses, dan dikelola dalam penyimpanan. Ini menyediakan abstraksi logis dari penyimpanan fisik (Silberschatz, Galvin & Gagne, 2018). Fungsi utama sistem berkas meliputi:

   a. Organisasi Data:
      Sistem berkas mengatur data dalam struktur hierarkis, biasanya menggunakan konsep direktori dan subdirektori (Tanenbaum & Bos, 2015).
   b. Naming:
      Sistem berkas menyediakan skema penamaan untuk berkas dan direktori, memungkinkan pengguna dan

program untuk mereferensikan data dengan nama yang mudah diingat (Stallings, 2018).

c. Kontrol Akses:
   Sistem berkas menerapkan mekanisme keamanan untuk mengontrol akses ke berkas dan direktori, biasanya melalui sistem permission (Silberschatz, Galvin & Gagne, 2018).
d. Manajemen Ruang Disk:
   Sistem berkas bertanggung jawab untuk alokasi dan dealokasi ruang disk, serta tracking ruang yang tersedia dan yang digunakan (Tanenbaum & Bos, 2015).
e. Journaling:
   Beberapa sistem berkas modern mengimplementasikan journaling untuk menjaga integritas data dalam kasus kegagalan sistem (Love, 2010).

Beberapa contoh sistem berkas yang umum digunakan:

a. ext4: Sistem berkas default untuk banyak distribusi Linux (Love, 2010).
b. NTFS: Sistem berkas standar untuk Windows (Russinovich, Solomon & Ionescu, 2012).
c. FAT32 dan exFAT: Sering digunakan untuk perangkat penyimpanan portabel (Carrier, 2005).

5. *Device Drivers*

Device drivers adalah komponen perangkat lunak yang memungkinkan sistem operasi dan aplikasi untuk berinteraksi dengan perangkat keras tanpa perlu mengetahui detail spesifik tentang perangkat tersebut (Egan & Teixeira, 2013). Fungsi utama device drivers meliputi:

a. Abstraksi Perangkat Keras:
   Drivers menyediakan antarmuka standar untuk sistem operasi, menyembunyikan kompleksitas perangkat keras spesifik (Corbet, Rubini & Kroah-Hartman, 2005).
b. Manajemen Sumber Daya:
   Drivers mengelola sumber daya perangkat, seperti memori I/O dan interrupt (Love, 2010).

c. Penanganan Error:
   Drivers bertanggung jawab untuk mendeteksi dan menangani kesalahan perangkat (Egan & Teixeira, 2013).
d. Optimasi Kinerja:
   Drivers sering mengimplementasikan teknik seperti buffering dan caching untuk meningkatkan kinerja (Corbet, Rubini & Kroah-Hartman, 2005).

Jenis-jenis device drivers:

a. *Character Device Drivers*: Untuk perangkat yang diakses sebagai aliran byte, seperti terminal (Love, 2010).
b. *Block Device Drivers*: Untuk perangkat yang diakses dalam blok data, seperti hard disk (Corbet, Rubini & Kroah-Hartman, 2005).
c. Network Device Drivers: Untuk perangkat jaringan seperti kartu jaringan (Benvenuti, 2006).

6. Subsistem Jaringan

Subsistem jaringan adalah komponen sistem operasi yang menangani komunikasi data melalui jaringan komputer. Ini menyediakan layanan dan protokol yang memungkinkan aplikasi untuk berkomunikasi melalui jaringan (Kurose & Ross, 2017). Fungsi utama subsistem jaringan meliputi:

a. Implementasi Protokol:
   Subsistem jaringan mengimplementasikan berbagai protokol jaringan seperti TCP/IP, UDP, HTTP, dan lainnya (Stevens, Fenner & Rudoff, 2004).
b. Manajemen Koneksi:
   Menangani pembentukan, pemeliharaan, dan pemutusan koneksi jaringan (Kurose & Ross, 2017).
c. Buffering dan Queueing:
   Mengelola buffer untuk data yang diterima dan dikirim, serta mengatur antrian paket (Benvenuti, 2006).

d. Routing:
   Menentukan rute terbaik untuk pengiriman paket data dalam jaringan (Tanenbaum & Wetherall, 2011).
e. Keamanan Jaringan:
   Mengimplementasikan mekanisme keamanan seperti firewall dan enkripsi (Stallings & Brown, 2018).

Komponen utama subsistem jaringan:

a. *Socket Interface*: Menyediakan API untuk aplikasi untuk mengakses layanan jaringan (Stevens, Fenner & Rudoff, 2004).
b. *Protokol Stack*: Implementasi dari protokol jaringan seperti TCP/IP (Kurose & Ross, 2017).
c. *Network Device Drivers*: Menghubungkan stack protokol dengan perangkat keras jaringan (Benvenuti, 2006).

7. Manajemen Keamanan

   Manajemen keamanan adalah komponen kritis dalam sistem operasi modern yang bertanggung jawab untuk melindungi sistem dan data dari akses yang tidak sah, serangan, dan ancaman keamanan lainnya (Stallings & Brown, 2018). Fungsi utama manajemen keamanan meliputi:

   a. Autentikasi:
      Memverifikasi identitas pengguna atau proses yang mencoba mengakses sistem atau sumber daya (Anderson, 2020).
   b. Otorisasi:
      Menentukan hak akses pengguna atau proses terhadap sumber daya sistem (Stallings & Brown, 2018).
   c. Enkripsi:
      Melindungi data dengan mengubahnya menjadi format yang tidak dapat dibaca tanpa kunci dekripsi yang tepat (Ferguson, Schneier & Kohno, 2010).
   d. Audit:
      Melacak dan mencatat aktivitas sistem untuk deteksi dan analisis keamanan (Anderson, 2020).

e. Isolasi Proses:
   Memastikan bahwa proses tidak dapat mengakses atau memodifikasi memori atau sumber daya proses lain tanpa izin (Tanenbaum & Bos, 2015).

8. Manajemen Daya

   Manajemen daya adalah komponen sistem operasi yang bertanggung jawab untuk mengoptimalkan penggunaan energi, terutama penting untuk perangkat mobile dan sistem embedded (Venkatesh & Chandrasekaran, 2019). Fungsi utama manajemen daya meliputi:

   a. *Dynamic Voltage and Frequency Scaling* (DVFS):
      Menyesuaikan voltase dan frekuensi CPU berdasarkan beban kerja untuk menghemat energi (Pallipadi & Starikovskiy, 2006).
   b. Power States Management:
      Mengelola berbagai state daya perangkat, seperti sleep, hibernate, dan active (Rao & Kenneth, 2015).
   c. *Device Power Management*:
      Mematikan atau mengurangi daya perangkat yang tidak digunakan (Venkatesh & Chandrasekaran, 2019).
   d. Thermal Management:
      Memonitor dan mengelola suhu sistem untuk mencegah overheating dan mengoptimalkan kinerja (Rao & Kenneth, 2015).
   e. Battery Life Optimization:
      Mengimplementasikan strategi untuk memperpanjang masa pakai baterai pada perangkat mobile (Pallipadi & Starikovskiy, 2006).

2.2 Tinjauan Tentang Arsitektur Monolitik, Mikrokernel, dan Hibrida

1. Arsitektur Monolitik

   Arsitektur monolitik adalah desain tradisional di mana seluruh sistem operasi berjalan sebagai satu program besar dalam mode kernel (Tanenbaum & Bos, 2015). Karakteristik utama:

- Semua layanan sistem operasi berjalan dalam ruang kernel.
- Komunikasi antar komponen sangat efisien.
- Performa tinggi karena fungsi-fungsi inti terhubung langsung.

Kelebihan:

- Kinerja tinggi karena overhead komunikasi rendah.
- Desain dan implementasi relatif sederhana.

Kekurangan:

- Sulit untuk dipelihara dan dikembangkan karena ukurannya yang besar.
- Kerusakan pada satu komponen dapat mempengaruhi seluruh sistem.

  Contoh: Unix tradisional, Linux kernel (Silberschatz, Galvin & Gagne, 2018).

  "Sistem operasi monolitik mengintegrasikan semua layanan ke dalam kernel. Semua komponen sistem beroperasi dalam mode kernel, berbagi ruang alamat yang sama" (Stallings, 2018, p.72).

2. Arsitektur Mikrokernel

   Mikrokernel adalah pendekatan di mana hanya fungsi-fungsi inti yang berjalan dalam mode kernel, sementara layanan lainnya berjalan sebagai proses di ruang pengguna (Liedtke, 1995). Karakteristik utama:

   - Kernel kecil yang hanya menyediakan layanan dasar.
   - Sebagian besar layanan sistem operasi berjalan sebagai proses pengguna.
   - Komunikasi antar komponen menggunakan passing pesan.

   Kelebihan:

   - Lebih mudah diperluas dan dipelihara.
   - Keamanan dan keandalan yang lebih baik karena isolasi komponen.

Kekurangan:

- Overhead kinerja karena komunikasi antar proses yang sering.
- Desain dan implementasi lebih kompleks.
  Contoh: MINIX, QNX (Tanenbaum & Woodhull, 2006).
  "Mikrokernel memindahkan sebanyak mungkin dari kernel ke ruang pengguna. Tujuannya adalah memiliki kernel kecil yang menyediakan mekanisme minimal" (Tanenbaum & Bos, 2015, p.65).

3. Arsitektur Hibrida

   Arsitektur hibrida menggabungkan elemen-elemen dari arsitektur monolitik dan mikrokernel (Stallings, 2018). Karakteristik utama:

   - Beberapa layanan berjalan dalam mode kernel untuk kinerja.
   - Layanan lain berjalan sebagai proses pengguna untuk fleksibilitas.

   Kelebihan:

   - Keseimbangan antara kinerja dan modularitas.
   - Fleksibilitas dalam penempatan layanan.

   Kekurangan:

   - Kompleksitas desain meningkat.
   - Potensi overhead karena komunikasi antar ruang kernel dan pengguna.

Contoh: Windows NT, macOS (Russinovich, Solomon & Ionescu, 2012).

"Sistem operasi hibrida mencoba mengambil keuntungan dari kedua pendekatan [monolitik dan mikrokernel]. Sebagian kode berjalan dalam kernel mode, sementara layanan tingkat tinggi berjalan dalam mode pengguna" (Silberschatz, Galvin & Gagne, 2018, p.78).

2.3 Konsep dan Prinsip Desain yang Mendasari Pengembangan Sistem Operasi Modern

Konsep dan Prinsip Desain Sistem Operasi Modern

1. Modularitas
   Sistem operasi modern dirancang dengan pendekatan modular, memungkinkan pengembangan dan pemeliharaan yang lebih mudah (Tanenbaum & Bos, 2015).
   Karakteristik:
   - Setiap modul memiliki antarmuka yang terdefinisi dengan baik.
   - Modul dapat dikembangkan, diuji, dan dimodifikasi secara independen.

   "Modularitas memungkinkan pengembang untuk memodifikasi bagian sistem tanpa mempengaruhi keseluruhan, meningkatkan maintainability dan fleksibilitas" (Silberschatz, Galvin & Gagne, 2018, p.89).

2. Abstraksi
   Menyembunyikan kompleksitas perangkat keras dari pengguna dan aplikasi, menyediakan antarmuka yang konsisten (Silberschatz, Galvin & Gagne, 2018).
   Contoh:
   - Abstraksi file menyembunyikan detail penyimpanan fisik.
   - Abstraksi proses menyembunyikan detail manajemen CPU.

   "Abstraksi memungkinkan pengembang aplikasi untuk fokus pada fungsionalitas tingkat tinggi tanpa perlu memahami detail implementasi low-level" (Tanenbaum & Bos, 2015, p.43).

3. Konkurensi
   Mendukung eksekusi simultan dari banyak proses atau thread (Stallings, 2018).
   Tantangan:
   - Sinkronisasi

- Deadlock prevention
  "Concurrency adalah kunci untuk memanfaatkan sepenuhnya sistem multi-core modern" (Tanenbaum & Bos, 2015, p.97).

4. Persistensi
   Menyediakan penyimpanan jangka panjang dan mekanisme untuk mempertahankan informasi (Tanenbaum & Bos, 2015).
   Aspek penting:
   - Integritas data
   - Konsistensi
   - Keamanan data

   "Persistensi memastikan bahwa data tetap tersedia dan konsisten bahkan setelah sistem di-reboot atau mengalami kegagalan" (Stallings, 2018, p.489).

5. Keamanan dan Perlindungan
   Mengimplementasikan mekanisme untuk melindungi sumber daya sistem dan data pengguna (Anderson, 2020).
   Elemen kunci:
   - *Principle of least privilege*
   - *Defense in depth*
   - *Secure defaults*

   "*Security by Design* memastikan bahwa keamanan menjadi bagian integral dari arsitektur sistem, bukan sekadar lapisan tambahan" (Anderson, 2020, p.27).

6. Skalabilitas
   Kemampuan untuk menangani peningkatan beban kerja dan sumber daya (Silberschatz, Galvin & Gagne, 2018).

7. Virtualisasi
   Menyediakan abstraksi sumber daya fisik, memungkinkan beberapa sistem operasi berjalan pada satu perangkat keras (Tanenbaum & Bos, 2015).
   Aplikasi:

- Virtual machines
- Containers

"Virtualisasi meningkatkan utilisasi sumber daya dan fleksibilitas dalam penggunaan perangkat keras" (Silberschatz, Galvin & Gagne, 2018, p.712).

8. Efisiensi Energi
   Mengoptimalkan penggunaan daya, terutama penting untuk perangkat mobile (Venkatesh & Chandrasekaran, 2019).

DAFTAR PUSTAKA

Anderson, R., 2020. Security engineering: a guide to building dependable distributed systems. John Wiley & Sons.

Benvenuti, C., 2006. Understanding Linux network internals. O'Reilly Media, Inc.

Blum, R. and Bresnahan, C., 2015. Linux command line and shell scripting bible. John Wiley & Sons.

Carrier, B., 2005. File system forensic analysis. Addison-Wesley Professional.

Corbet, J., Rubini, A. and Kroah-Hartman, G., 2005. Linux device drivers. O'Reilly Media, Inc.

Egan, O. and Teixeira, S., 2013. Developing drivers with the Windows Driver Foundation. Microsoft Press.

Ferguson, N., Schneier, B. and Kohno, T., 2010. Cryptography engineering: design principles and practical applications. John Wiley & Sons.

Kurose, J.F. and Ross, K.W., 2017. Computer networking: a top-down approach. Pearson.

Liedtke, J., 1995. On micro-kernel construction. ACM SIGOPS Operating Systems Review, 29(5), pp.237-250.

Love, R., 2010. Linux kernel development. Pearson Education.

Pallipadi, V. and Starikovskiy, A., 2006. The ondemand governor. In Proceedings of the Linux Symposium (Vol. 2, pp. 215-230).

Rao, V.N. and Kenneth, Y.T., 2015. Embedded Systems Power Management. Newnes.

Robbins, A., 2016. Bash pocket reference: Help for power users and sys admins. O'Reilly Media, Inc.

Russinovich, M.E., Solomon, D.A. and Ionescu, A., 2012. Windows internals. Pearson Education.

Shotts, W., 2019. The Linux command line: a complete introduction. No Starch Press.

Silberschatz, A., Galvin, P.B. and Gagne, G., 2018. Operating system concepts. John Wiley & Sons.

Stallings, W., 2018. Operating systems: internals and design principles. Pearson.

Stevens, W.R., Fenner, B. and Rudoff, A.M., 2004. UNIX network programming. Addison-Wesley Professional.
Stevens, W.R. and Rago, S.A., 2013. Advanced programming in the UNIX environment. Addison-Wesley.
Tanenbaum, A.S. and Woodhull, A.S., 2006. Operating systems: design and implementation. Pearson.
Tanenbaum, A.S. and Wetherall, D., 2011. Computer networks. Pearson.
Tanenbaum, A.S. and Bos, H., 2015. Modern operating systems. Pearson.
Venkatesh, J. and Chandrasekaran, B., 2019. Electronic Design Automation for Integrated Circuits Handbook-2 Volume Set. CRC Press.

BAB 3
MANAJEMEN PROSES DAN THREAD

Oleh Lutfi Syafirullah

3.1 Pendahuluan

Proses dalam sistem komputer adalah suatu entitas yang merupakan instance dari sebuah program yang sedang berjalan. Ketika pengguna menjalankan suatu program, sistem operasi akan menciptakan sebuah proses untuk program tersebut. Proses ini memiliki sumber daya seperti memori, waktu CPU, dan ruang alamat yang diperlukan untuk menjalankan program tersebut. Proses bertanggung jawab untuk melakukan tugas-tugas yang diminta oleh pengguna atau sistem. Setelah proses selesai dieksekusi, hasil dari proses tersebut dapat dikembalikan kepada pengguna atau digunakan untuk melanjutkan operasi sistem yang lebih kompleks.

Keterkaitan antara proses dan sistem operasi terkait erat dengan cara sistem operasi menjalankan atau mengeksekusi proses tersebut. Terdapat dua cara utama di mana sistem operasi mengeksekusi proses:

1. **Batch System (Sistem Batch)**

   Dalam sistem batch, proses-proses dieksekusi secara berurutan dan berdasarkan antrian. Pengguna atau administrator sistem mengumpulkan berbagai program (*batch jobs*) dalam satu batch. Sistem operasi kemudian mengeksekusi batch ini tanpa interaksi langsung dengan pengguna. Setelah satu batch selesai dieksekusi, sistem akan mengeksekusi batch berikutnya.

   Contoh penggunaan sistem *batch* adalah pada proses pengolahan transaksi perbankan, pengolahan data besar, atau pemrosesan laporan.

2. ***Time-Sharing System*** **(Sistem Berbagi Waktu)**
   Dalam sistem *time-sharing* (atau disebut juga timesharing atau multitasking), sistem operasi membagi waktu CPU di antara berbagai proses yang berjalan secara simultan. Setiap proses diberikan sebagian kecil waktu untuk dieksekusi sebelum beralih ke proses lainnya. Proses ini berlangsung sangat cepat sehingga memberikan ilusi bahwa banyak proses berjalan secara bersamaan(Maya Utami Dewi, 2021).
   Pengguna dapat berinteraksi langsung dengan proses-proses ini melalui antarmuka pengguna, seperti terminal atau GUI (*Graphical User Interface*). Contoh sistem time-sharing adalah sistem operasi modern seperti Windows, macOS, dan Linux yang mendukung penggunaan multitasking dengan banyak aplikasi berjalan secara bersamaan.

3.2 Manajemen Proses

Setiap proses dalam sistem komputer membutuhkan sumber daya untuk menjalankan tugasnya. Dengan pengelolaan sumber daya yang efisien dan pengalokasian yang tepat oleh sistem operasi, sistem komputer dapat menjalankan banyak proses secara bersamaan dengan cara yang aman, efisien, dan tanpa konflik. Beberapa sumber daya yang dibutuhkan oleh sebuah proses dan bagaimana sistem operasi mengelola sumber daya tersebut:

1. **CPU Time** (waktu pemrosesan CPU): Setiap proses membutuhkan waktu CPU untuk menjalankan instruksi-instruksi programnya. CPU time dialokasikan kepada proses oleh sistem operasi. Proses dapat menggunakan waktu CPU ini untuk melakukan komputasi, pengolahan data, dan berbagai operasi lainnya yang memerlukan kecepatan proses tinggi.
2. **Memori** (ruang memori utama): Proses memerlukan memori untuk menyimpan kode program, data yang sedang diproses, variabel, dan struktur data lainnya. Sistem operasi mengalokasikan sejumlah memori untuk setiap proses saat proses tersebut dibuat atau dijalankan. Memori ini biasanya dikelola dalam bentuk virtual memory untuk efisiensi

penggunaan dan untuk memisahkan memori antara berbagai proses agar tidak saling mengganggu.

3. **Berkas-Berkas (**data yang disimpan di disk): Proses seringkali perlu mengakses berkas-berkas yang tersimpan di sistem, seperti berkas teks, gambar, basis data, dan lain sebagainya. Sistem operasi menyediakan mekanisme untuk proses agar bisa membaca, menulis, dan mengelola berkas-berkas ini. Alokasi sumber daya untuk berkas-berkas ini tergantung pada hak akses yang dimiliki oleh proses dan pengelolaan file system yang ditangani oleh sistem operasi.
4. **Perangkat I/O (**keyboard, mouse, dan perangkat lainnya): Proses juga dapat berinteraksi dengan perangkat I/O seperti printer, keyboard, mouse, jaringan, dan perangkat lainnya. Sistem operasi menyediakan abstraksi dan kontrol atas perangkat-perangkat ini. Sumber daya perangkat I/O dialokasikan untuk proses agar proses dapat berkomunikasi dengan perangkat dan melakukan operasi input-output yang diperlukan.

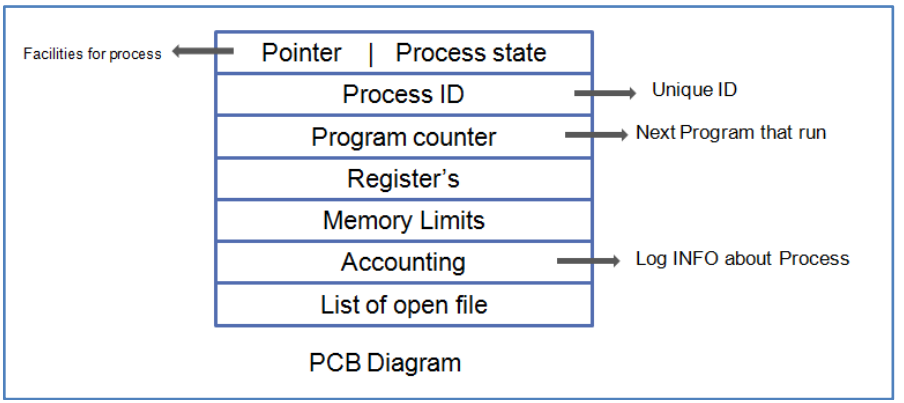
Ketika sebuah proses diciptakan atau sedang diproses oleh sistem operasi, sistem tersebut mengalokasikan sumber daya yang diperlukan agar proses dapat berjalan dengan baik. Misalnya, CPU time dialokasikan agar proses dapat dieksekusi, memori dialokasikan agar proses memiliki tempat untuk menyimpan data dan instruksi yang sedang diproses, dan perangkat I/O dihubungkan agar proses dapat berinteraksi dengan pengguna atau perangkat lain.

Ketika proses tersebut selesai dieksekusi atau berhenti dijalankan, sistem operasi akan mengambil kembali semua sumber daya yang telah dialokasikan untuk proses tersebut. Proses ini dikenal sebagai dealokasi sumber daya. Dealokasi ini penting karena memastikan bahwa sumber daya sistem digunakan secara efisien dan tidak ada sumber daya yang tetap terikat pada proses yang sudah tidak berjalan lagi. Misalnya, memori yang dialokasikan untuk proses akan kembali ke sistem agar bisa digunakan untuk proses lain yang sedang berjalan atau baru akan dibuat. Dengan cara ini, sistem operasi mengelola

sumber daya secara efisien, memungkinkan banyak proses untuk berjalan secara bersamaan dan tetap mempertahankan keseimbangan yang baik antara kinerja dan penggunaan sumber daya.

Ketika proses memasuki sistem, mereka dimasukkan ke dalam antrian input, sistem operasi memperhitungkan persyaratan memori dari setiap proses dan jumlah ruang memori yang tersedia. Ketika sebuah proses berakhir, ia melepaskan memorinya, yang kemudian dapat diisi oleh sistem operasi dengan proses lain dari antrian input(Abraham Silberschatz, Peter Baer Galvin, 2019).

3.2.1 Process Control Block



Gambar 3.1. Process Control Block (PCB)
(Sumber: Sistem Operasi dan Utility, Maya Utami Dewi, S.Kom., M.Kom.)

Setiap proses dalam sistem operasi direpresentasikan oleh sebuah *Process Control Block* (PCB) atau Control Block. PCB adalah struktur data penting yang menyimpan informasi lengkap tentang sebuah proses yang sedang berjalan atau siap untuk dieksekusi di dalam sistem computer(Maya Utami Dewi, 2021).

PCB ini menyimpan informasi penting tentang suatu proses yang sedang berjalan di system. Prioritas proses adalah nilai atau besaran yang digunakan oleh sistem operasi untuk menentukan urutan atau frekuensi eksekusi proses di dalam

sistem komputer. Secara umum, proses dengan prioritas yang lebih tinggi akan dijadwalkan untuk dieksekusi lebih sering atau lebih dulu dibandingkan dengan proses-proses yang memiliki prioritas lebih rendah. Prioritas proses sangat penting dalam pengaturan kinerja sistem operasi karena memungkinkan sistem untuk mengoptimalkan penggunaan sumber daya seperti CPU dan memori. PCB (*Process Control Block*) memuat informasi yang sangat penting terkait dengan sebuah proses dalam sistem operasi, yaitu:

1. **Process ID or PID**

   Process ID (PID) adalah identifikasi unik berupa bilangan bulat yang diberikan kepada setiap proses dalam sistem operasi. PID ini berfungsi sebagai nomor identitas yang membedakan satu proses dengan proses lainnya dalam sistem. Setiap proses, baik yang sedang berjalan, sedang menunggu, atau dalam status lainnya, memiliki PID yang unik. PID memungkinkan sistem operasi untuk secara unik mengidentifikasi dan merujuk setiap proses dalam sistem. Dengan menggunakan PID, sistem operasi dapat melakukan berbagai operasi seperti penjadwalan, pengalokasian sumber daya, manajemen memori, dan manajemen inter-proses yang efisien dan efektif.Tahap -tahap PID dalam PCB (*Process Control Block*):

   a. **Penciptaan Proses (*Creation*)**

      Saat proses baru diciptakan oleh sistem operasi, sebuah PID diberikan ke proses tersebut. PCB pada tahap ini mencatat PID proses baru bersama dengan informasi awal lainnya seperti status awal, prioritas, dan lokasi memori.

   b. **Penjadwalan dan Eksekusi Proses (*Scheduling and Execution*)**

      Proses yang siap dieksekusi (*ready*) atau sedang dieksekusi (*running*) memiliki PID yang digunakan oleh sistem operasi untuk mengidentifikasi dan mengelola eksekusi proses. PCB menyimpan PID bersama dengan informasi konteks register CPU yang terkait dengan proses yang sedang dieksekusi, memungkinkan sistem

operasi untuk memantau dan mengelola jalannya proses ini di CPU.

c. **Menunggu Kejadian (*Waiting for Events*)**
   Ketika suatu proses menunggu kejadian tertentu seperti I/O selesai, proses tersebut tetap memiliki PID yang sama. PCB pada tahap ini mungkin mencatat status "waiting" dan alasan mengapa proses tersebut menunggu.
d. **Berhenti atau Selesai (*Halted or Finished*)**
   Ketika proses selesai dieksekusi atau dihentikan secara paksa, PID tetap tercatat dalam PCB. Informasi tentang berhentinya proses dan pembebasan sumber daya mungkin juga dicatat dalam PCB.

2. ***Process Stage***

Salah satu komponen kunci dalam PCB adalah informasi tentang tahapan atau status (*process stage*) dari proses tersebut. Tahapan atau status ini menggambarkan kondisi saat ini dari proses tersebut dalam sistem operasi. Berikut adalah penjelasan mengenai bagaimana PCB menyimpan informasi mengenai tahapan proses:

a. ***Ready*** **(Siap)**
   Proses dalam status r*eady* siap untuk dieksekusi oleh CPU. PCB akan mencatat bahwa proses ini telah diinisialisasi dan sudah siap untuk mendapatkan alokasi CPU berikutnya.
b. ***Waiting*** **(Menunggu)**
   Proses yang sedang menunggu berarti proses tersebut sedang menunggu terjadinya suatu kondisi atau kejadian tertentu, seperti I/O selesai atau sinyal dari proses lain. PCB akan mencatat jenis kejadian yang sedang ditunggu oleh proses ini.
c. ***Running*** **(Menjalankan)**
   Proses dalam status running sedang aktif dieksekusi oleh CPU. PCB akan mencatat informasi detail mengenai proses ini, termasuk nilai-nilai register CPU saat ini dan instruksi yang sedang dieksekusi.

d. *Exit* (Keluar)

Proses yang telah menyelesaikan eksekusi atau dihentikan secara paksa akan berada dalam status exit. PCB pada tahapan ini mungkin akan mencatat informasi terkait dengan alasan berhentinya proses dan proses pembebasan sumber daya yang digunakan.

3. Process Privileges

Salah satu komponen penting yang terdapat dalam PCB adalah hak akses atau priviledges (*Process Privileges*) yang dimiliki oleh proses tersebut terhadap berbagai sumber daya seperti memori dan perangkat. Berikut adalah beberapa informasi yang PCB mungkin menyimpan terkait dengan Process Privileges:

a. Hak Akses terhadap Memori

PCB dapat mencatat informasi tentang segmen memori yang dialokasikan untuk proses tersebut. Ini termasuk alamat awal dari segmen kode (*code segment*), segmen data (*data segment*), dan segmen tumpukan (*stack segment*). Informasi ini memungkinkan sistem operasi untuk memantau dan mengatur akses proses terhadap memori, serta mencegah proses dari melakukan akses ilegal atau tak sah.

b. Hak Akses terhadap Perangkat (*Devices*)

Proses mungkin memerlukan akses khusus terhadap perangkat keras seperti printer, disk drive, atau jaringan.P CB dapat mencatat izin khusus atau hak akses yang diberikan kepada proses terhadap perangkat-perangkat ini. Misalnya, proses tertentu mungkin memiliki izin untuk menulis ke file tertentu tetapi tidak untuk membaca file lainnya.

c. Hak Akses terhadap Sumber Daya Sistem Lainnya

PCB juga dapat mencatat izin khusus terhadap sumber daya sistem lainnya seperti file-file khusus, direktori, atau hak akses administratif tertentu. Informasi ini membantu dalam menjaga keamanan sistem operasi dengan memastikan bahwa setiap proses hanya

memiliki akses yang sesuai dengan kebutuhan dan izin yang diberikan.

d. Hak Akses ke Jaringan (*Network Access*)
Jika proses memerlukan akses ke jaringan untuk berkomunikasi, PCB dapat menyimpan informasi mengenai hak akses jaringan yang diberikan kepada proses tersebut. Ini termasuk informasi tentang protokol jaringan yang dapat digunakan, alamat jaringan yang diizinkan, dan sebagainya.

4. Pointer

Penggunaan "Pointer" untuk menunjukkan lokasi proses induk (*parent process*) mungkin tidak umum atau tidak ditemukan dalam PCB secara langsung. Namun, ada beberapa cara di mana informasi terkait proses induk dapat diwakilkan dalam PCB atau struktur terkait lainnya dalam sistem operasi. Berikut adalah beberapa cara umum di mana informasi tentang proses induk dapat dikaitkan atau diwakilkan:

a. Hierarki Proses (*Process Hierarchy*)
Dalam sistem operasi yang mendukung hierarki proses, setiap proses biasanya memiliki identifikasi terkait dengan proses induknya (*parent process*). Informasi ini dapat diwakilkan dengan menggunakan PID (Process ID) dari proses induk. Sebuah PCB dapat mencatat PID dari proses induk sebagai bagian dari informasi yang terkait dengan hubungan hierarkis antara proses-proses dalam sistem.

b. Struktur Data Tambahan
Meskipun tidak secara langsung dalam PCB, sistem operasi dapat menggunakan struktur data tambahan seperti tabel entri proses atau struktur yang mirip untuk menyimpan hubungan antar proses. Pointer atau referensi ke entri proses induk dapat disimpan di dalam struktur data ini, memungkinkan sistem operasi untuk dengan cepat merujuk ke proses induk dari sebuah proses tertentu.

c. **Manajemen Tumpang Tindih (*Overlapping Management*)**
   Dalam sistem operasi modern, seperti yang mendukung *multi-threading* atau *multi-processing*, konsep "*parent process*" mungkin kurang relevan karena banyak proses atau thread dapat dimiliki oleh satu entitas induk. PCB atau struktur data yang terkait dapat mencatat informasi tentang entitas yang lebih besar (misalnya, sebuah grup atau job) daripada sekadar proses induk tunggal.

5. **Program Counter**
   Salah satu komponen utama dari PCB adalah Program Counter (PC), yang merupakan register khusus dalam CPU yang menunjukkan alamat dari instruksi berikutnya yang akan dieksekusi oleh proses. Beberapa hal mengenai Program Counter:
   a. **Fungsi Program Counter**
      Program Counter adalah register CPU yang menyimpan alamat dari instruksi berikutnya yang akan dieksekusi oleh proses. Dalam PCB, nilai Program Counter untuk proses tertentu disimpan agar sistem operasi dapat memantau dan mengendalikan alur eksekusi proses tersebut.
   b. **Penyimpanan Nilai Program Counter dalam PCB**
      Ketika proses beralih antar proses atau ketika proses berhenti sementara (misalnya saat melakukan I/O), nilai Program Counter saat itu disimpan dalam PCB. Informasi ini memungkinkan sistem operasi untuk kembali ke instruksi yang belum selesai dieksekusi ketika proses dilanjutkan.
   c. **Manajemen Eksekusi Proses**
      Dengan menyimpan nilai Program Counter dalam PCB, sistem operasi dapat melanjutkan eksekusi proses dari titik terakhir ketika proses di-schedule kembali. Ini juga memungkinkan sistem operasi untuk melakukan manajemen sumber daya yang efisien dan melakukan kontrol terhadap alur eksekusi proses yang berjalan.

d. Sinkronisasi antara CPU dan Memori

Program Counter penting dalam memastikan sinkronisasi antara CPU dan memori, karena menunjukkan instruksi mana yang sedang atau akan dieksekusi berikutnya. PCB mencatat nilai Program Counter untuk setiap proses, memungkinkan sistem operasi untuk melakukan penjadwalan yang tepat dan efisien antara proses-proses yang sedang berjalan.

6. CPU Registers

Salah satu komponen utama dari PCB adalah CPU Registers (register CPU), yang menyimpan nilai-nilai register CPU yang relevan untuk proses tersebut. Beberapa hal mengenai CPU Registers:

a. Fungsi CPU Registers

Registers CPU adalah lokasi penyimpanan internal yang sangat cepat di dalam CPU. Registers ini digunakan untuk menyimpan data yang akan diolah atau hasil dari operasi-operasi yang sedang berjalan. Sebelum eksekusi program, registers CPU diatur dengan nilai-nilai awal yang diperlukan untuk memulai eksekusi proses, seperti nilai Program Counter (PC) untuk menunjukkan instruksi berikutnya.

b. Penyimpanan Nilai Registers dalam PCB

PCB menyimpan snapshot atau salinan dari nilai-nilai registers CPU yang relevan untuk proses tersebut pada saat proses berhenti atau beralih. Informasi ini mencakup nilai dari registers umum seperti register data (data registers), register address (*address registers*), register status (*status registers*), dan register khusus lainnya yang relevan untuk eksekusi proses.

c. Pengelolaan Konteks Proses

Dengan menyimpan nilai registers CPU dalam PCB, sistem operasi dapat mengembalikan konteks eksekusi proses dengan benar ketika proses dilanjutkan atau beralih kembali ke proses tersebut. Ini penting untuk menjaga konsistensi dan keamanan eksekusi proses serta memastikan bahwa proses dapat melanjutkan

pekerjaannya dari titik terakhir eksekusi tanpa kehilangan data atau status.

d. Sinkronisasi dengan Eksekusi Program

Registers CPU berhubungan erat dengan eksekusi program karena menyimpan data yang sedang diproses oleh proses. PCB mencatat nilai-nilai registers CPU untuk setiap proses, memungkinkan sistem operasi untuk melakukan penjadwalan yang tepat dan efisien antara proses-proses yang sedang berjalan.

7. Scheduling Information

Salah satu komponen utama dari PCB adalah informasi penjadwalan (*scheduling information*), yang berisi detail tentang bagaimana proses akan dipilih dan dijadwalkan untuk dieksekusi oleh CPU berdasarkan algoritma penjadwalan yang diterapkan dalam sistem operasi. Berikut adalah penjelasan lebih lanjut mengenai Scheduling Information dalam PCB:

a. Algoritma Penjadwalan (*Scheduling Algorithms*)

Sistem operasi menggunakan berbagai jenis algoritma penjadwalan untuk menentukan urutan eksekusi proses di dalam CPU. Contoh algoritma penjadwalan termasuk FCFS (*First-Come, First-Served*), SJF (*Shortest Job First*), Round Robin, Priority Scheduling, dan lain-lain. Scheduling Information dalam PCB mencakup parameter atau nilai-nilai yang diperlukan oleh sistem operasi untuk menerapkan algoritma penjadwalan tertentu. Misalnya, untuk algoritma Priority Scheduling, PCB akan menyimpan nilai prioritas dari proses tersebut.

b. Prioritas Eksekusi (*Execution Priority*)

Informasi penjadwalan juga mencakup prioritas eksekusi proses. Setiap proses diberi nilai prioritas yang menentukan urutan eksekusinya relatif terhadap proses-proses lain yang sedang berjalan. PCB akan menyimpan nilai prioritas ini sehingga sistem operasi dapat memutuskan proses mana yang harus dieksekusi terlebih dahulu ketika CPU tersedia.

c. **Status Siap (*Ready Status*)**
   Scheduling Information mencatat status siap (*ready*) dari proses. Proses yang siap untuk dieksekusi oleh CPU akan memiliki status siap dalam PCB. Informasi ini membantu sistem operasi dalam mengetahui proses mana yang dapat segera diberikan waktu eksekusi selanjutnya.
d. **Data Historis dan Statistik**
   Beberapa sistem operasi mungkin juga mencatat data historis atau statistik terkait dengan eksekusi proses sebelumnya, seperti waktu eksekusi yang telah berlalu, jumlah siklus CPU yang digunakan, atau informasi kinerja lainnya. Informasi ini digunakan untuk meningkatkan keputusan penjadwalan di masa mendatang dan untuk analisis kinerja sistem secara keseluruhan.

8. **Memory Management Information**
   Salah satu komponen utama dari PCB adalah informasi manajemen memori (Memory Management Information, MIM), yang mencakup detail-detail penting yang diperlukan oleh sistem operasi untuk mengelola dan mengatur alokasi memori bagi proses-proses yang sedang berjalan. Berikut adalah penjelasan lebih lanjut mengenai *Memory Management Information* dalam PCB:
   a. ***Page Table* (Tabel Halaman)**
      Dalam sistem manajemen memori virtual, proses mengakses memori fisik melalui tabel halaman. PCB menyimpan informasi mengenai *page table* yang berhubungan dengan proses tersebut. Page table memberikan pemetaan antara alamat virtual (yang digunakan oleh proses) dan alamat fisik di memori. Informasi ini memungkinkan sistem operasi untuk menerjemahkan referensi memori dari proses ke alamat fisik yang sesuai.
   b. **Batas Memori (*Memory Limits*)**
      Setiap proses memiliki batas maksimum memori yang dapat digunakan. Informasi ini disimpan dalam PCB

untuk memastikan bahwa proses tidak melebihi alokasi memori yang telah ditetapkan. Batas memori juga dapat berhubungan dengan segmentasi memori, di mana setiap segment (seperti kode, data, stack) memiliki batas yang didefinisikan.

c. *Segment Table* (Tabel Segmen)

Beberapa sistem operasi menggunakan segmentasi memori sebagai teknik untuk mengatur ruang alamat proses. Informasi tentang tabel segmen disimpan dalam PCB untuk proses-proses yang menggunakan metode segmentasi. Segment table mengatur alamat dan ukuran dari masing-masing segmen dalam ruang alamat virtual proses.

d. Manajemen Memori Virtual (*Virtual Memory Management*)

PCB juga mungkin menyimpan informasi terkait dengan manajemen memori virtual, seperti penggunaan swap space (ruang tukar) untuk memori yang digunakan oleh proses yang tidak aktif saat ini. Informasi ini membantu dalam pengaturan alokasi memori yang efisien dan optimal di dalam sistem operasi.

e. Pembebasan Sumber Daya Memori (*Memory Resource Deallocation*)

Ketika proses selesai dieksekusi atau dihentikan, informasi tentang penggunaan memori oleh proses tersebut dicatat dalam PCB. PCB memungkinkan sistem operasi untuk membebaskan atau mengalokasikan kembali sumber daya memori yang telah digunakan oleh proses yang telah berakhir.

9. Accounting Information

Salah satu komponen utama dari PCB adalah informasi akuntansi (*Accounting Information*), yang mencatat statistik dan data terkait dengan eksekusi proses. Berikut adalah penjelasan lebih lanjut mengenai Accounting Information dalam PCB:

a. **Waktu Eksekusi (*Execution Time*)**
   PCB menyimpan informasi tentang waktu yang dibutuhkan oleh proses untuk menyelesaikan eksekusi. Ini mencakup waktu mulai (*start time*) proses, waktu selesai (*finish time*), dan total waktu yang dihabiskan dalam eksekusi. Informasi ini membantu dalam memantau kinerja proses dan menganalisis efisiensi waktu dalam sistem operasi.
b. **Execution ID**
   Execution ID atau ID eksekusi adalah identifikasi unik yang diberikan kepada setiap eksekusi proses dalam sistem. Ini dapat mencakup nomor eksekusi tertentu atau informasi identifikasi lainnya yang membantu dalam melacak dan mengelola eksekusi proses secara individu. PCB dapat menyimpan Execution ID untuk merujuk kembali ke riwayat atau statistik khusus eksekusi proses tertentu.
c. **Batas (*Limits*)**
   Informasi batas dalam *Accounting Information* mencakup batas-batas tertentu yang terkait dengan proses, seperti batas memori yang diizinkan, batas waktu CPU yang diberikan, atau batas sumber daya lainnya. Sistem operasi menggunakan informasi ini untuk memantau kepatuhan proses terhadap batasan yang telah ditetapkan dan untuk mengambil tindakan jika proses melebihi batas-batas ini.
d. **Statistik Kinerja**
   Selain waktu eksekusi, Accounting Information juga dapat mencatat statistik kinerja lainnya seperti jumlah instruksi dieksekusi, jumlah interupsi atau pengecualian yang terjadi, atau penggunaan sumber daya seperti I/O operations. Statistik ini membantu dalam analisis kinerja proses dan sistem secara keseluruhan, serta memungkinkan untuk identifikasi potensi masalah atau perbaikan yang diperlukan.

e. **Manajemen Sumber Daya**
   PCB juga dapat menyimpan informasi tentang penggunaan sumber daya yang dilakukan oleh proses, seperti penggunaan memori, alokasi file sistem, atau penggunaan perangkat keras lainnya. Informasi ini penting untuk manajemen sumber daya yang efektif dan pengoptimalan penggunaan sumber daya di dalam sistem operasi.

10. I/O Status

Salah satu komponen dari PCB adalah informasi status I/O (I/O Status), yang mencatat detail tentang sumber daya input/output yang dapat digunakan oleh proses. Berikut adalah penjelasan lebih lanjut mengenai I/O Status dalam PCB:

a. **Daftar Sumber Daya I/O**
   PCB menyimpan daftar atau informasi tentang sumber daya input/output yang dapat diakses oleh proses tertentu. Ini termasuk perangkat keras seperti keyboard, mouse, printer, disk drive, dan lain sebagainya. Informasi ini memungkinkan sistem operasi untuk mengelola akses dan penggunaan sumber daya I/O oleh proses sesuai dengan izin yang telah ditetapkan.

b. **Status Operasi I/O**
   PCB mencatat status dari operasi I/O yang sedang dilakukan oleh proses. Misalnya, apakah proses sedang menunggu respon dari operasi I/O tertentu, sedang dalam proses membaca atau menulis ke perangkat, atau telah selesai melakukan operasi I/O. Informasi ini membantu sistem operasi dalam mengatur penjadwalan proses yang melibatkan operasi I/O, serta untuk menangani kondisi-kondisi khusus seperti blokade atau kesalahan I/O.

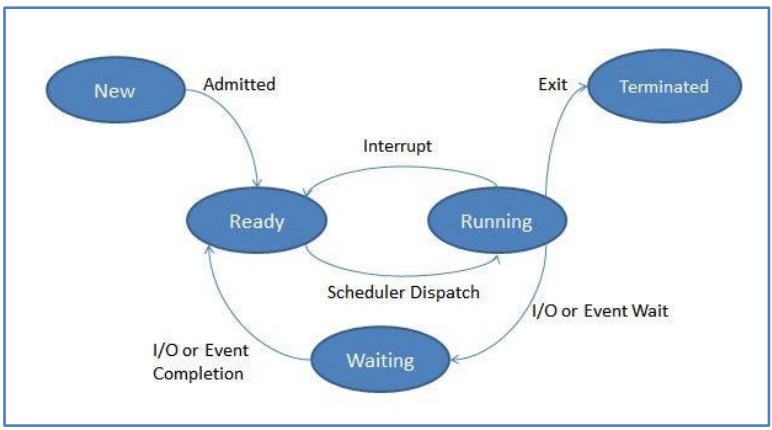
c. **Buffering dan Cacheing**
   PCB juga dapat menyimpan informasi tentang buffering dan caching yang terkait dengan operasi I/O. Buffering memungkinkan data untuk disimpan sementara dalam

memori sebelum dikirim atau setelah diterima dari perangkat I/O. Informasi caching menentukan apakah data telah di-cache untuk akses cepat oleh proses atau apakah data harus diambil langsung dari perangkat I/O.

d. **Pengaturan Prioritas dan Manajemen Konflik**
PCB juga dapat mencatat pengaturan prioritas terkait dengan operasi I/O, memberikan prioritas tinggi untuk operasi tertentu jika diperlukan. Sistem operasi menggunakan informasi ini untuk mengelola konflik akses ke sumber daya I/O, memastikan bahwa proses yang membutuhkan akses cepat dan prioritas tinggi dapat dilayani dengan segera.

e. **Informasi Sinyal dan Kejadian**
Terkadang, PCB juga dapat mencatat informasi tentang sinyal atau kejadian yang terkait dengan operasi I/O, seperti sinyal interrupt yang terjadi ketika operasi I/O telah selesai atau ketika terjadi kesalahan pada perangkat.

3.2.2 Status Proses



Gambar 3.2. Diagram Status Proses
(Sumber: Sistem Operasi, Abas Ali Pangera, M.Kom.&Dony Ariyus)

Dalam sistem operasi, status sebuah proses mencerminkan keadaan atau aktivitas dari proses tersebut selama siklus hidupnya.

Berikut adalah beberapa status umum yang mungkin dimiliki sebuah proses (Abas Ali Pangera, 2005):

1. ***New* (Baru)**
   Proses dalam status ini baru saja dibuat atau diinisialisasi oleh sistem operasi tetapi belum siap untuk dieksekusi. Pada tahap ini, sistem operasi biasanya melakukan inisialisasi awal seperti alokasi sumber daya dan mempersiapkan lingkungan untuk proses.
2. ***Ready* (Siap)**
   Proses yang telah siap untuk dieksekusi oleh CPU disebut sebagai proses yang siap. Proses dalam status ini telah mendapatkan semua sumber daya yang diperlukan dan menunggu giliran untuk dieksekusi oleh CPU.
3. ***Running* (Berjalan)**
   Proses dalam status ini sedang dieksekusi oleh CPU. CPU sedang menjalankan instruksi-instruksi dari proses tersebut secara aktif.
4. ***Waiting* (Menunggu)**
   Proses yang sedang menunggu suatu kejadian atau kondisi tertentu untuk dapat melanjutkan eksekusinya disebut proses yang sedang menunggu. Contohnya, proses dapat menunggu input dari pengguna, respon dari I/O operation, atau sinyal dari proses lain.
5. ***Terminated* (Dihentikan)**
   Proses yang telah menyelesaikan eksekusi utamanya atau secara manual dihentikan oleh sistem operasi disebut proses yang telah dihentikan. Pada tahap ini, sistem operasi membersihkan sumber daya yang digunakan oleh proses dan menyimpan statistik akhir eksekusi.

3.2.3 Pembuatan Proses

Dalam sistem operasi, konsep induk proses (*parent process*) dan anak proses (*child process*) sangat penting dalam hierarki proses yang dibentuk(Maya Utami Dewi, 2021).

Ketika suatu proses menciptakan proses baru melalui sistem pemanggilan pembuatan proses, proses asalnya disebut induk atau parent process, sedangkan proses yang baru dibuat

disebut anak atau child process. Induk proses adalah proses yang membuat atau menciptakan proses baru melalui sistem pemanggilan pembuatan proses (seperti *'fork()'* dalam UNIX/Linux). Proses induk bertanggung jawab atas proses baru yang diciptakan, dan dapat terus menjalankan tugasnya atau menunggu anak prosesnya menyelesaikan tugasnya terlebih dahulu. Dengan konsep ini, sistem operasi dapat mengatur dan mengelola eksekusi proses secara hierarkis dan efisien, mengizinkan aplikasi untuk mengorganisir tugas-tugasnya dalam struktur yang terstruktur dan dapat diatur dengan baik.

Secara umum, manajemen sumber daya dan operasi terhadap proses sangat penting untuk memastikan efisiensi, keamanan, dan kestabilan sistem secara keseluruhan. Sistem operasi bertanggung jawab untuk mengalokasikan memori kepada setiap proses sesuai dengan kebutuhannya. Ketika suatu proses menciptakan subproses, sistem operasi akan mengalokasikan ruang memori yang cukup untuk subproses tersebut. Sistem operasi juga memastikan bahwa proses-proses tidak melebihi batas memori yang telah ditetapkan agar tidak terjadi pengisian memori yang berlebihan (*memory overcommitment*). Sistem operasi melakukan penjadwalan (*scheduling*) proses untuk menentukan proses mana yang akan dieksekusi oleh CPU pada waktu tertentu. Ketika suatu proses meminta CPU untuk mengeksekusi instruksi-instruksinya, sistem operasi akan mengatur akses CPU tersebut untuk memastikan bahwa proses mendapatkan waktu eksekusi yang wajar sesuai dengan kebutuhannya. Operasi-operasi terhadap proses juga mencakup manajemen akses terhadap berkas-berkas di sistem. Sistem operasi mengizinkan proses untuk membuka, menulis, membaca, dan menutup berkas-berkas secara aman sesuai dengan hak akses yang telah ditetapkan (*permissions*). Proses dapat berinteraksi dengan perangkat I/O seperti disk drive, printer, atau jaringan. Sistem operasi menyediakan akses yang terkontrol terhadap perangkat I/O ini untuk memastikan bahwa setiap proses dapat menggunakan perangkat tersebut sesuai dengan kebutuhan tanpa mengganggu proses lain atau menyebabkan konflik. Sistem operasi

menangani kontrol akses terhadap sumber daya yang dimiliki proses, termasuk memastikan bahwa proses hanya dapat mengakses sumber daya yang diizinkan sesuai dengan kebijakan keamanan yang telah ditetapkan. Misalnya, sistem operasi dapat menerapkan mekanisme hak akses untuk membatasi akses proses terhadap data atau sumber daya lainnya yang sensitif. Ketika suatu proses menciptakan subproses, sistem operasi dapat membatasi atau mengatur sumber daya yang dapat diakses oleh subproses tersebut. Ini termasuk memastikan bahwa subproses tidak menghabiskan terlalu banyak sumber daya sehingga mengganggu kinerja sistem secara keseluruhan.

3.2.4 Terminasi Proses

Proses terminasi dalam sistem operasi merupakan proses di mana sistem operasi menghentikan eksekusi suatu proses yang sedang berjalan. Saat proses ini terminasi, beberapa langkah penting dilakukan oleh sistem operasi untuk membersihkan semua sumber daya yang telah dialokasikan kepada proses tersebut (Maya Utami Dewi, 2021).

Sebuah proses berakhir ketika proses tersebut menyelesaikan eksekusi semua pernyataan dalam programnya dan kemudian meminta sistem operasi untuk menghapusnya dengan menggunakan sistem pemanggilan exit. Sistem operasi menghapus entri PCB (*Process Control Block*) dari daftar proses yang sedang berjalan. Hal ini membebaskan sumber daya yang digunakan oleh proses, seperti memori dan deskriptor berkas. Sebelum proses keluar, proses dapat mengembalikan data atau keluaran kepada proses induknya. Proses ini dilakukan melalui sistem pemanggilan wait, di mana proses induk menunggu hasil atau status dari proses anaknya. Sistem operasi mengelola penghapusan semua sumber daya yang dialokasikan kepada proses tersebut. Ini termasuk membebaskan memori fisik dan virtual yang digunakan oleh proses, menutup file yang dibuka oleh proses, dan menyelesaikan operasi I/O yang masih tertunda. Setelah proses keluar, sistem operasi mengirimkan sinyal status kepada proses induk untuk memberitahu apakah proses anaknya telah berakhir dengan sukses atau ada

kesalahan. Informasi ini membantu proses induk dalam menentukan tindakan selanjutnya, seperti menangani hasil dari proses anak atau melanjutkan eksekusi. Proses terminasi penting dilakukan untuk menjaga efisiensi penggunaan sumber daya oleh sistem operasi, dengan membersihkan sumber daya secara sistematis setelah sebuah proses selesai atau dihentikan, sistem operasi dapat mengalokasikan kembali sumber daya yang berharga ini kepada proses-proses lain yang sedang berjalan atau yang akan dieksekusi selanjutnya.

Langkah-langkah dalam terminasi proses:

1. **Memori Fisik dan Virtual**
   Sistem operasi akan menghapus entri yang terkait dengan proses dari tabel memori fisik dan tabel memori virtual. Ini mencakup membebaskan ruang memori fisik di RAM yang sebelumnya digunakan oleh proses dan menghapus entri yang menunjukkan lokasi memori virtual yang dialokasikan untuk proses tersebut.
2. **Penyimpanan I/O**
   Jika proses telah membuka berkas atau melakukan operasi I/O (*Input/Output*), sistem operasi akan menangani penyimpanan kembali semua sumber daya yang terlibat. Ini termasuk menutup berkas yang mungkin masih terbuka oleh proses, sehingga sumber daya tersebut dapat digunakan oleh proses lain di masa mendatang.
3. **Penyimpanan Informasi Terkait Proses**
   Informasi-informasi yang terkait dengan proses, seperti struktur data yang menyimpan status proses, informasi hak akses, dan informasi administratif lainnya, akan dibersihkan dan dikembalikan ke sistem operasi. Hal ini dilakukan agar sumber daya yang sudah tidak diperlukan tidak mengganggu proses-proses lain yang berjalan di sistem.

Sistem operasi akan menghapus entri yang terkait dengan proses dari tabel memori fisik dan tabel memori virtual. Ini mencakup membebaskan ruang memori fisik di RAM yang sebelumnya digunakan oleh proses dan menghapus entri yang

menunjukkan lokasi memori virtual yang dialokasikan untuk proses tersebut.

3.2.5 Komunikasi dan Hubungan Antar Proses

1. **Memory Sharing System**

   Berbagi memori atau shared memory system merupakan salah satu mekanisme komunikasi antar proses dalam sistem operasi modern. Konsepnya adalah dengan mengalokasikan suatu area memori tertentu yang dapat diakses (dibagikan) oleh beberapa proses secara bersamaan.

   Prinsip Kerja Shared Memory System:

   a. **Alokasi Memori Bersama**

      Pembuat program menentukan suatu area memori tertentu yang akan digunakan sebagai shared memory. Area ini biasanya dialokasikan oleh salah satu proses (biasanya proses pembuat atau proses khusus yang bertugas mengelola shared memory) dan kemudian dibagikan kepada proses-proses lain yang memerlukan akses.

   b. **Alamat dan Ukuran Alokasi**

      Alamat dan ukuran dari area memori yang dialokasikan sebagai shared memory ditentukan dalam implementasi program. Proses-proses yang ingin berkomunikasi melalui shared memory harus mengetahui alamat dan ukuran tersebut agar bisa mengakses dan memanipulasi data yang ada di dalamnya.

   c. **Manajemen Akses**

      Sistem operasi bertanggung jawab untuk mengatur akses ke shared memory agar proses-proses yang menggunakan shared memory dapat beroperasi secara aman dan efisien. Hal ini bisa termasuk mekanisme penguncian (*locking*) untuk memastikan bahwa hanya satu proses yang dapat mengubah data pada saat tertentu (misalnya menggunakan mutex atau semafor).

d. Komunikasi Efektif

Dengan menggunakan shared memory, proses-proses yang berbagi data dapat berkomunikasi secara langsung tanpa perlu melakukan komunikasi melalui mekanisme IPC (*Inter-Process Communication*) yang lain, seperti pesan antar-proses (message passing). Hal ini bisa mengurangi overhead yang terjadi dalam proses komunikasi dan memungkinkan proses-proses untuk berbagi data dengan kecepatan tinggi.

Penggunaan *shared memory* umumnya ditemukan dalam aplikasi yang membutuhkan komunikasi cepat dan efisien antar proses, seperti dalam sistem basis data yang memungkinkan proses-proses server untuk berbagi cache data, dalam aplikasi multimedia untuk berbagi data video atau audio yang sedang diproses, atau dalam aplikasi yang memerlukan komputasi paralel dengan memori bersama. Dengan demikian, *Shared Memory System* adalah salah satu cara efektif untuk memfasilitasi komunikasi dan koordinasi antar proses dalam lingkungan sistem operasi modern, dengan memanfaatkan kecepatan akses memori dan mengurangi overhead yang terkait dengan komunikasi antar-proses.

Keuntungan *shared memory* system menyebabkan data dapat diakses langsung dari memori dan proses komunikasi dapat dilakukan dengan cepat, mengurangi overhead yang terkait dengan mekanisme IPC lainnya serta konsepnya relatif sederhana untuk diimplementasikan dan dimengerti.

Kekurangan *shared memory system* diperlukan manajemen sinkronisasi yang cermat untuk menghindari masalah race condition atau deadlock, jika jumlah proses yang mengaksesnya semakin banyak diperlukan pengelolaan yang hati-hati terhadap konflik akses dan penggunaan sumber daya sehingga diperlukan perhatian khusus terhadap keamanan data yang dibagikan untuk mencegah akses yang tidak sah.

2. *Message Sharing System*

Message Sharing System (atau sering disebut juga dengan *Message Passing System*) adalah salah satu mekanisme yang digunakan dalam sistem operasi untuk memungkinkan komunikasi antar proses yang berjalan secara independen. Sistem ini memungkinkan proses-proses yang berbeda untuk bertukar informasi atau data melalui pengiriman pesan dari satu proses ke proses lainnya.

Prinsip Kerja *Message Sharing System*:

a. Pengiriman Pesan

Dalam *Message Sharing System*, proses-proses dapat saling berkomunikasi dengan mengirim dan menerima pesan. Pesan ini dapat berisi data, sinyal, atau instruksi tertentu yang diperlukan oleh proses penerima untuk melakukan tugasnya.

b. Mekanisme Sistem

Sistem operasi menyediakan mekanisme yang memungkinkan proses untuk mengirim pesan ke proses lain. Hal ini bisa dilakukan melalui pemanggilan sistem khusus (*system call*) yang disediakan oleh sistem operasi, seperti send() untuk mengirim pesan dan receive() untuk menerima pesan.

c. Buffering

Sistem operasi umumnya menyediakan buffer atau antrian (*queue*) untuk menyimpan pesan yang sedang dikirim atau menunggu untuk diterima. Buffer ini dapat dikelola oleh sistem operasi untuk memastikan pesan dikirim dan diterima dengan aman.

d. Synchronization

Message Sharing System juga dapat menyediakan mekanisme untuk sinkronisasi antar proses, misalnya dengan menggunakan semafor atau mekanisme lain untuk memastikan bahwa pesan hanya dikirim saat penerima siap menerima dan sebaliknya.

3.2.6 Sinkronisasi

Sinkronisasi merupakan proses pengaturan atau koordinasi jalannya beberapa proses agar bekerja secara teratur dan terkoordinasi. Hal ini penting karena dalam lingkungan komputasi modern, sering kali terdapat kebutuhan untuk memastikan bahwa proses-proses yang berjalan dapat saling berinteraksi, berbagi sumber daya, dan menghindari situasi yang tidak diinginkan seperti deadlock(Nidhom, 2020).

Salah satu tujuan utama dari sinkronisasi adalah untuk mencegah terjadinya inkonsistensi data ketika beberapa proses atau thread berusaha mengakses dan memanipulasi data yang sama secara bersamaan. Konsep ini dikenal dengan istilah mutual exclusion (mutex). Dengan menerapkan mutex sebelum mengakses data bersama, proses dapat memastikan bahwa operasi baca/tulis dilakukan secara berurutan, sehingga data yang diakses tidak rusak atau tidak konsisten(Nidhom, 2020).

Komunikasi antar proses membutuhkan penggunaan subroutine atau fungsi primitif khusus yang memungkinkan proses untuk mengirim dan menerima data. Implementasi dari primitif ini dapat berbeda-beda tergantung pada desain sistem yang digunakan. Salah satu perbedaan utama dalam implementasi primitif tersebut adalah apakah operasi pengiriman atau penerimaan pesan dilakukan secara bloking atau non-bloking, yang juga dikenal sebagai sinkron atau asinkron(Maya Utami Dewi, 2021).

Kondisi dalam keadaan sinkron:

1. *Blocking send*.
   Pengiriman pesan bloking berarti proses pengirim akan terblokir (*blocked*) sampai pesan yang dikirimnya benar-benar diterima oleh proses penerima/berhasil disalin ke dalam buffer yang disediakan oleh sistem operasi untuk proses penerima.
2. *Blocking receive*.
   Proses penerima akan terblokir sampai pesan yang dikirimkan oleh proses pengirim tiba dan tersedia untuk diproses.

Dalam implementasi nyata, kombinasi dari kedua pendekatan ini sering digunakan untuk memenuhi kebutuhan komunikasi yang berbeda dalam sistem yang kompleks dan terdistribusi. Pemilihan yang tepat tergantung pada faktor seperti keamanan data, kecepatan respons, dan efisiensi penggunaan sumber daya.

Kondisi dalam keadaan asinkron:

a. Non-blocking send
   Proses pengirim akan melanjutkan eksekusi segera setelah operasi send dimulai, tanpa menunggu konfirmasi bahwa pesan telah diterima oleh proses penerima. Setelah memanggil operasi send, proses pengirim dapat melanjutkan eksekusi segera tanpa menunggu pesan sampai benar-benar diterima oleh proses penerima.
b. Non-blocking receive
   proses penerima akan melanjutkan eksekusi segera setelah operasi receive dimulai, tanpa menunggu konfirmasi bahwa pesan telah dikirim oleh proses pengirim. Setelah memanggil operasi receive, proses penerima dapat melanjutkan eksekusi segera tanpa menunggu sampai pesan tiba di buffer.

3.2.7 Penundaan

Penundaan (*suspend*) merujuk pada operasi yang dilakukan terhadap proses-proses yang sedang berjalan untuk sementara waktu menghentikan eksekusi mereka. Praktik ini dilakukan dengan beragam cara dan tujuan, tergantung pada kebutuhan sistem dan strategi manajemen beban. Salah satu tujuan utama dari penundaan adalah untuk mengelola dan mereduksi beban sistem operasi. Pada saat beban puncak atau ketika sistem sedang mengalami tekanan berlebihan, penundaan dapat digunakan untuk menangani sementara proses-proses yang tidak kritis atau yang dapat dijalankan kemudian. Dalam beberapa sistem operasi, penundaan digunakan untuk mengatur prioritas eksekusi antara proses-proses yang berjalan. Proses-proses dengan prioritas rendah

atau tidak kritis dapat ditangguhkan sementara untuk memberikan lebih banyak sumber daya kepada proses-proses dengan prioritas yang lebih tinggi. enundaan juga dapat digunakan sebagai bagian dari strategi manajemen sumber daya. Misalnya, sistem operasi dapat menunda proses-proses yang mengonsumsi banyak CPU atau memori untuk sementara waktu untuk mencegah kelebihan penggunaan sumber daya yang dapat mengganggu kinerja keseluruhan sistem. Dengan menggunakan penundaan secara efektif, sistem operasi dapat mereduksi beban, mengoptimalkan kinerja sumber daya, dan memprioritaskan tugas sesuai dengan kebutuhan sistem dan pengguna. Proses membebaskan sumber daya biasanya terjadi saat proses tersebut selesai atau dihentikan secara paksa (misalnya, karena error atau interupsi). Membebaskan sumber daya dengan cepat dan efisien penting untuk memaksimalkan penggunaan sumber daya sistem secara keseluruhan. Resuming (pengaktifan kembali) adalah proses mengaktifkan kembali (menjalankan kembali) suatu proses dari titik (instruksi) dimana proses tersebut ditangguhkan atau dihentikan sebelumnya. Konsep ini terkait dengan manajemen siklus hidup proses, di mana proses dapat berada dalam berbagai keadaan seperti jalan (*running*), siap (*ready*), tertunda (*blocked*), atau siap dihentikan (*ready suspended*). Membebaskan sumber daya memungkinkan penggunaan kembali sumber daya yang berharga seperti memori dan perangkat I/O, sementara resuming proses memungkinkan proses untuk melanjutkan eksekusi dari titik dimana proses tersebut ditangguhkan sebelumnya. Keduanya penting untuk optimalisasi penggunaan sumber daya dan responsivitas sistem secara keseluruhan.

3.2.8 Buffering

Buffering merujuk pada penggunaan antrian atau buffer sementara untuk menyimpan data atau pesan yang sedang ditransfer antara proses-proses atau perangkat. Konsep buffering sangat penting dalam memastikan komunikasi yang efisien dan dapat diandalkan antara entitas yang berbeda dalam sistem operasi. Sistem operasi menyediakan berbagai mekanisme dan strategi

untuk implementasi buffering, tergantung pada kebutuhan aplikasi dan desain sistem. Implementasi buffering yang efektif dan efisien membantu meningkatkan kinerja keseluruhan sistem operasi dan memastikan pengiriman data yang dapat diandalkan antar proses dan perangkat. Dengan buffering, proses dapat melanjutkan eksekusi tanpa harus menunggu proses I/O selesai. Ini mengurangi waktu yang dihabiskan dalam status menunggu. Buffering membantu memastikan bahwa data yang dikirimkan atau diterima dijamin akan tersedia untuk diproses oleh penerima atau perangkat tujuan. Buffer dapat menyimpan data sementara jika penerima tidak siap untuk menerima atau jika perangkat I/O tidak dapat mengonsumsi data dengan kecepatan yang sama seperti yang dikirim. Buffering memungkinkan sistem operasi untuk mengatur penggunaan sumber daya secara lebih efisien, terutama dalam kasus I/O yang memerlukan alokasi memori sementara untuk menampung data. Antrian dapat dibedakan berdasarkan kapasitasnya, yaitu:

1. Kapasitas Nol
   Antrian dengan kapasitas nol biasanya tidak memiliki kapasitas penyimpanan internal yang signifikan. Ini berarti bahwa tidak ada data yang dapat disimpan dalam antrian ini antara produsen (*producer*) dan konsumen *(consumer*) jika konsumen belum siap untuk mengonsumsinya.
2. Kapasitas Terbatas
   Antrian dengan kapasitas terbatas memiliki ukuran maksimum yang ditentukan untuk menyimpan data sementara antara produsen dan konsumen. Jumlah maksimum item yang dapat disimpan dalam antrian ini ditentukan pada saat inisialisasi antrian.
3. Kapasitas Tak Terbatas
   Antrian dengan kapasitas tak terbatas tidak memiliki batas atas pada jumlah elemen yang dapat disimpan di dalamnya. Antrian ini dapat tumbuh seiring waktu sejauh memori yang tersedia memungkinkan.

3.2.9 Manajemen Proses Sistem Operasi Windows

Pada sistem operasi Windows, konsep proses dan thread memiliki hubungan yang erat dan penting untuk memahami cara kerja dari sebuah aplikasi atau sistem yang sedang berjalan. Setiap proses memiliki ruang alamat virtualnya sendiri, sumber daya sistem seperti memori, dan daftar kontrol yang mengatur sumber daya yang digunakan oleh proses tersebut. Proses dalam Windows dibuat dengan memanggil fungsi API *CreateProcess*(). Fungsi ini digunakan untuk membuat proses baru berdasarkan executable file (.exe). Ketika *CreateProcess*() dipanggil, Windows akan membuat ruang alamat baru untuk proses tersebut, mengalokasikan memori dan sumber daya yang diperlukan, dan mulai menjalankan kode dari file executable yang ditentukan. Proses dalam Windows dapat memiliki satu atau lebih thread. Thread adalah unit eksekusi yang lebih kecil dalam konteks proses yang berbagi ruang alamat dan sumber daya dengan proses utama. Thread di dalam proses dapat dibuat menggunakan fungsi API CreateThread().

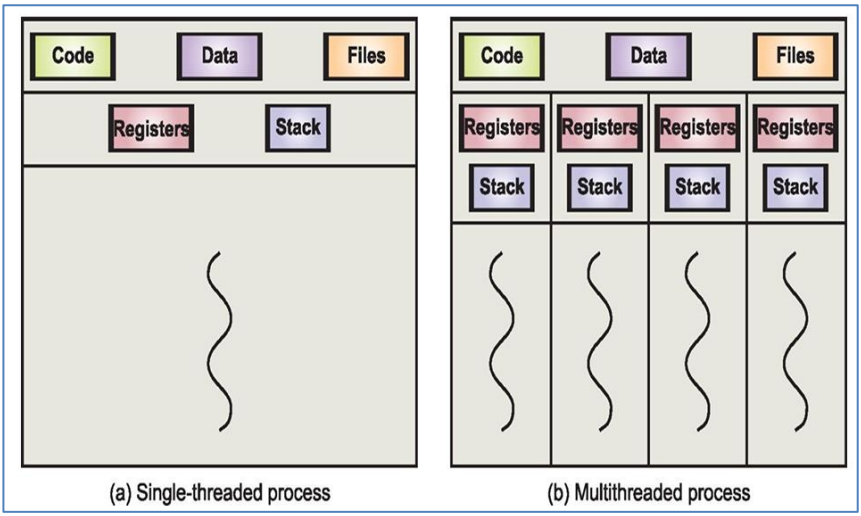
3.3 Thread

Suatu proses pada dasarnya adalah sebuah program yang sedang dijalankan oleh sistem operasi komputer. Biasanya, ketika sebuah program dijalankan, mungkin melibatkan beberapa proses untuk mencapai konkurensi, yaitu kemampuan untuk menjalankan tugas secara bersamaan. Namun, mengelola beberapa proses memerlukan biaya tambahan karena harus beralih di antara proses tersebut. Overhead ini mencakup penyimpanan status proses yang sedang berjalan (pengalihan konteks) dan memuat status proses baru ke dalam memori. Untuk mengatasi overhead yang terkait dengan pengelolaan banyak proses, pendekatan alternatif adalah dengan menggunakan thread. Thread adalah unit eksekusi yang lebih ringan dibandingkan dengan proses.

3.3.1 *Thread Concept*

Thread merupakan unit dasar dari penggunaan CPU yang berisi beberapa elemen penting seperti Thread_ID, program counter, register set, dan stack. Sebuah thread berbagi code section,

data section, dan sumber daya sistem operasi dengan *thread-thread* lain yang dimiliki oleh proses yang sama. Istilah "*lightweight process*" sering digunakan untuk menggambarkan thread karena thread memiliki overhead yang lebih kecil dibandingkan dengan proses tradisional atau heavyweight process.



Gambar 3.3. Model Single Thread dan Multi Thread
(Sumber: Operating System Second Edition, Rohit Khurana)

Dalam sistem operasi modern, konsep multithreading telah dikembangkan untuk memungkinkan sebuah proses untuk menjalankan lebih dari satu thread eksekusi secara bersamaan di dalam satu ruang alamat yang sama. Ini memberikan beberapa keuntungan yang signifikan, terutama dalam hal meningkatkan responsifitas aplikasi dan memanfaatkan sumber daya CPU secara lebih efisien.

Setiap proses biasanya memiliki setidaknya satu thread eksekusi yang mengeksekusi instruksi-instruksi dari program tersebut. Thread ini adalah unit dasar eksekusi di dalam proses yang berjalan di atas CPU. Sebagai contoh, ketika sebuah proses menjalankan aplikasi *Word Processor*, proses tersebut mengalokasikan satu thread tunggal untuk menjalankan semua instruksi yang diperlukan oleh program tersebut. Thread ini

mengatur operasi dari membuka dokumen hingga menanggapi input pengguna, dan menampilkan hasil pada layar. Kendali oleh thread tunggal berarti bahwa pada satu waktu tertentu, proses hanya dapat melakukan satu tugas. Meskipun beberapa operasi mungkin terjadi secara bersamaan (seperti menulis teks sambil melakukan pengecekan ejaan), semuanya dikelola oleh satu thread yang dieksekusi secara berurutan(Khurana, 2015).

Thread dan proses memiliki perbedaan dalam hal hubungan dengan proses induk, pengelolaan ruang alamat, dan penggunaan sumber daya saat diciptakan, berikut perbedaannya:

1. Thread merupakan subset dari sebuah proses dan bergantung pada proses tersebut, sedangkan proses bisa bersifat independent
2. Setiap proses anak memiliki ruang alamat yang terpisah dari proses induknya, sementara thread yang berasal dari proses yang sama membagi ruang alamat dari proses tersebut.
3. Pembuatan thread umumnya membutuhkan lebih sedikit sumber daya dibandingkan pembuatan proses. Ketika sebuah thread dibuat, sistem operasi mengalokasikan struktur data yang berisi kumpulan register, stack, dan prioritas dari thread tersebut. Sebaliknya, ketika sebuah proses dibuat, blok kontrol proses (PCB) dialokasikan, yang merupakan struktur data besar. PCB mencakup peta memori, daftar file terbuka, dan variabel lingkungan. Pengalokasian dan pengelolaan peta memori biasanya merupakan aktivitas yang memakan waktu lebih lama.

Keuntungan penggunaan thread adalah sebagai berikut:

1. Peningkatan Kecepatan Komputasi
   Pada sistem dengan satu prosesor, sebuah proses dapat dieksekusi dengan cepat dengan membuat beberapa thread di dalamnya dan menjalankannya secara semiparalel (dengan cepat beralih CPU di antara beberapa thread). Ini menghasilkan peningkatan kecepatan komputasi karena beberapa thread dapat bekerja secara bersamaan.
2. Ekonomis

Pembuatan thread lebih ekonomis dibandingkan pembuatan proses. Ketika sebuah proses dibuat, memori dan sumber daya perlu dialokasikan untuknya. Namun, thread menggunakan sumber daya dari proses tempat mereka berada, sehingga tidak perlu alokasi memori dan sumber daya saat pembuatan thread.

3. Komunikasi yang Efisien
   Karena thread-thread dalam sebuah proses menggunakan ruang alamat yang sama, komunikasi di antara mereka dapat dilakukan melalui berbagi memori. Hal ini menghilangkan kebutuhan untuk sistem panggilan (*system calls*) yang memerlukan overhead tambahan.
4. Pemanfaatan yang Tepat dari Arsitektur Multiprosesor
   Pada sistem multiprosesor, thread lebih bermanfaat dibandingkan proses. Banyak thread dari satu proses dapat dijalankan pada CPU yang berbeda secara bersamaan, mencapai paralelisme nyata. Sebaliknya, satu proses hanya dapat berjalan pada satu CPU saja, terlepas dari jumlah CPU yang tersedia.
5. Responsif
   Pada proses interaktif, waktu respons menjadi kriteria kinerja utama. Jika proses tersebut multithreaded, bagian dari proses (*thread*) dapat tetap berjalan meskipun bagian lain dari proses tersebut terblokir. Akibatnya, responsifitas proses terhadap pengguna meningkat.

Ada dua metode umum untuk mengimplementasikan thread, yaitu kernel-level threads dan user-level threads(Khurana, 2015).

1. Kernel-level Threads
   Kernel-level threads diimplementasikan langsung oleh kernel sistem operasi. Kernel bertanggung jawab untuk menciptakan, menjadwalkan, dan mengelola thread dalam ruang kernel. Kernel memelihara tabel thread tambahan selain tabel proses yang menyimpan program counter, stack pointer, register, status, dan informasi penting lainnya dari setiap thread dalam sistem. Ketika sebuah proses ingin membuat atau mengakhiri sebuah thread baru, proses

tersebut melakukan pemanggilan sistem (system call) ke kernel. Sebagai respons, kernel akan membuat atau mengakhiri thread dengan mengubah tabel thread.

Keuntungan:

- Pada lingkungan multiprosesor, beberapa kernel-level threads yang berasal dari satu proses dapat dijadwalkan untuk berjalan secara bersamaan pada CPU yang berbeda, sehingga meningkatkan kecepatan komputasi.
- Karena thread dikelola langsung oleh kernel, jika satu thread mengeluarkan pemanggilan sistem yang memblokirnya, kernel dapat memilih thread lain untuk dijalankan baik dari proses yang sama maupun proses lain.

Kerugian:

- Biaya untuk menciptakan dan menghancurkan thread di kernel relatif lebih besar dibandingkan dengan user-level threads.
- Pengalihan (*switching*) antar thread oleh kernel memerlukan *overhead* yang dapat mempengaruhi kinerja sistem secara keseluruhan.

2. User-level Threads

User-level threads diimplementasikan oleh sebuah pustaka thread yang terkait dengan kode dari suatu proses. Pustaka thread ini menyediakan dukungan untuk menciptakan, menjadwalkan, dan mengelola thread secara mandiri di ruang pengguna.

Keuntungan:

- Biaya untuk menciptakan dan menghancurkan thread dalam ruang pengguna jauh lebih rendah dibandingkan dengan kernel-level threads.
- Pengelolaan thread di dalam ruang pengguna memungkinkan fleksibilitas yang lebih besar dalam manajemen thread oleh aplikasi.

Kerugian:

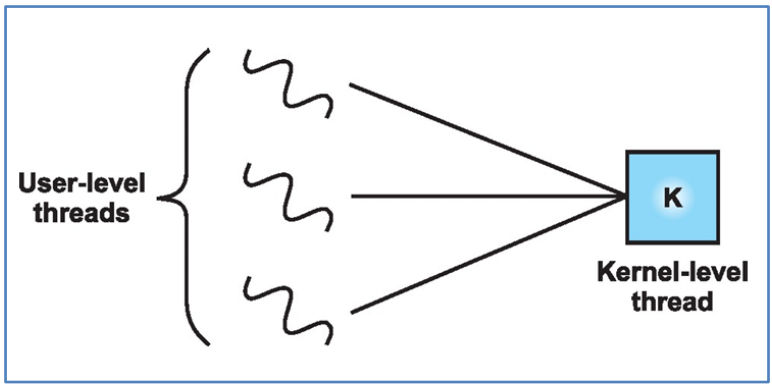
- User-level threads tidak dapat dijadwalkan secara langsung pada CPU yang berbeda dalam sistem

multiprosesor karena kernel tidak mengetahui tentang thread-thread ini secara langsung.
- Jika satu thread dalam proses melakukan pemanggilan sistem yang memblokir, semua thread dalam proses tersebut menjadi terblokir.

3.3.2 Multithreading Models

Ada beberapa model multithreading yang umum digunakan dalam sistem komputer modern, yang menggabungkan baik thread-level pengguna (*user-level threads*) maupun kernel-level threads, serta hubungan antara keduanya.

1. *Many-to-One* Model



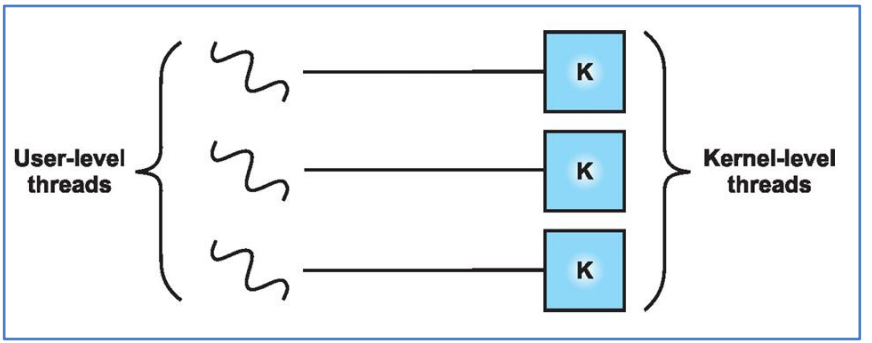
Gambar 3.4. Model Many-to-One (M:1)
(Sumber: Operating System Second Edition, Rohit Khurana)

Dalam model ini, banyak user-level threads dipetakan ke satu kernel-level thread. Artinya, semua pengelolaan dan penjadwalan thread dilakukan di ruang pengguna oleh pustaka thread (*thread library*), dan hanya satu kernel-level thread yang mengelola proses dan dapat dijadwalkan oleh kernel. Jika salah satu user-level thread melakukan pemanggilan sistem yang memblokir, seluruh proses akan diblokir.

Kelebihan:

Penjadwalan thread cepat dan efisien di ruang pengguna. Overhead yang lebih rendah karena hanya satu thread yang berhubungan dengan kernel.
Kekurangan:
Tidak ada paralelisme di tingkat kernel; hanya satu thread yang dapat dieksekusi pada satu waktu tertentu.

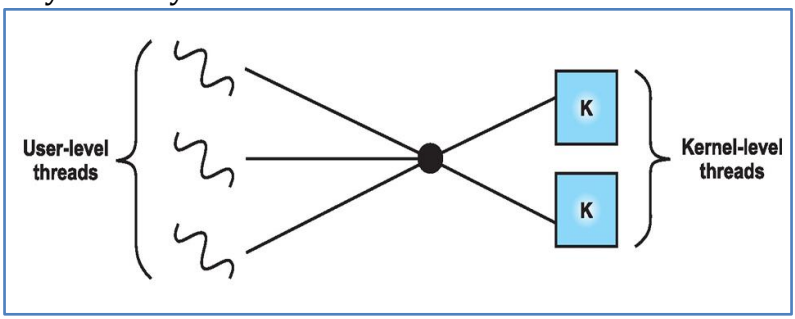
2. One-to-One Model



Gambar 3.5. Model One-to-One (1:1)
(Sumber: Operating System Second Edition, Rohit Khurana)

Dalam model ini, setiap user-level thread dipetakan langsung ke satu kernel-level thread. Kernel mengelola dan menjadwalkan setiap kernel-level thread secara terpisah, memungkinkan paralelisme yang lebih baik dalam lingkungan multiprosesor.
Kelebihan:
Paralelisme yang lebih baik karena setiap user-level thread dapat dieksekusi secara independen pada kernel-level. Responsif terhadap pemanggilan sistem dari masing-masing thread.
Kekurangan:
Biaya overhead lebih tinggi karena setiap user-level thread membutuhkan pengelolaan yang lebih intensif di kernel.

3. Many-to-Many Model



Gambar 3.6. Model Many-to-Many (M:M)
(Sumber: Operating System Second Edition, Rohit Khurana)

Dalam model ini, beberapa user-level threads dipetakan ke beberapa kernel-level threads. Ini memungkinkan fleksibilitas yang lebih besar dalam manajemen thread. Kernel dapat menangani penjadwalan dan sinkronisasi antara user-level threads dan kernel-level threads secara bersamaan.
Kelebihan:
Fleksibilitas dalam manajemen thread dengan dukungan untuk banyak thread pada banyak prosesor. Kombinasi antara kecepatan dan efisiensi dari user-level threads dengan manfaat paralelisme dari kernel-level threads.
Kekurangan:
Memerlukan implementasi yang kompleks dan membutuhkan manajemen sinkronisasi yang hati-hati antara user dan kernel-level threads.

3.3.3 Threading Issues

Terdapat beberapa isu dalam menjalankan program multithreaded, beberapa masalah atau isu sebagai berikut:

1. *System Calls Fork*() dan Exec()
   Ketika sebuah proses multithreaded memanggil sistem panggilan fork(), terdapat pertanyaan mengenai apakah thread yang baru dibuat (*child process*) harus mengandung semua thread dari proses asal (*parent process*) atau hanya thread yang melakukan pemanggilan fork() (thread

pemanggil). Setelah proses child dibuat dengan fork(), child process dapat menggunakan sistem panggilan exec() untuk memuat program lain ke dalam ruang alamatnya. Sistem panggilan exec() akan mengganti kode dari child process dengan program baru yang diinginkan.
Contoh Penerapan:
Misalkan ada sebuah aplikasi web server multithreaded. Ketika aplikasi tersebut menerima permintaan untuk menangani suatu request, bisa terjadi bahwa proses baru (*child process*) perlu menyalin hanya thread yang melakukan pemanggilan (jika akan menjalankan exec() untuk menangani request tersebut). Namun, jika child process tersebut hanya perlu menyalin semua thread dari proses asal untuk menjalankan tugas yang sama seperti proses asalnya, tanpa perlu menggantikan program, maka diperlukan versi fork() yang menyalin semua thread.

2. Pembatalan Thread
   Thread cancellation atau pembatalan thread adalah prosedur untuk menghentikan sebuah thread sebelum selesai dieksekusi. Thread yang akan dibatalkan disebut sebagai target thread. Ada dua cara umum untuk melakukan thread cancellation:
   a. Pembatalan *Asynchronous* (Pembatalan Tidak Bersamaan)
      Dalam pembatalan asynchronous, target thread dihentikan secara langsung setelah thread lain menunjukkan bahwa pembatalan diperlukan. Sistem operasi mungkin membebaskan sumber daya yang dialokasikan untuk thread yang dibatalkan, namun tidak selalu semua sumber daya dibebaskan secara sistemik. Hal ini dapat menyebabkan sistem dalam keadaan tidak konsisten. Thread dibatalkan secara instan setelah permintaan pembatalan. Risiko meninggalkan sumber daya sistem yang tidak dibebaskan sepenuhnya.

b. Pembatalan Deferred (Pembatalan Ditunda)
   Dalam pembatalan deferred, target thread tidak dihentikan secara langsung. Sebaliknya, thread ini secara teratur memeriksa apakah ia harus dibatalkan atau tidak. Thread target memiliki kesempatan untuk mengakhiri dirinya sendiri dengan cara yang teratur dan terorganisir. Thread target memeriksa pada interval tertentu apakah ia harus dihentikan dan memiliki kontrol untuk menentukan saat yang aman untuk dibatalkan. Pembatalan deferred memastikan konsistensi sistem dengan menentukan titik-titik di dalam kode thread di mana pembatalan dapat dilakukan dengan aman.

3. Data khusus thread
   Data khusus thread (thread-specific data) adalah data yang dimiliki oleh masing-masing thread secara terpisah dari thread-thread lain dalam sebuah proses. Meskipun thread-thread dalam proses berbagi data umum dari proses tersebut, ada situasi di mana suatu thread memerlukan salinan data tertentu yang berbeda dan independen dari thread lainnya.
   Karakteristik Data Khusus Thread:
   a. Independen untuk Setiap Thread
      Data khusus thread memungkinkan setiap thread memiliki salinan data yang berbeda dan independen dari thread-thread lainnya. Ini memungkinkan thread untuk menyimpan informasi yang spesifik untuk tugas atau konteks yang sedang dijalankan.
   b. Akses Bersamaan Aman
      Meskipun data khusus thread dimiliki secara terpisah, mekanisme akses aman perlu diimplementasikan untuk memastikan bahwa setiap thread dapat mengakses dan memodifikasi data khususnya tanpa konflik dengan thread lain.
   c. Pengelolaan Data
      Implementasi data khusus thread biasanya melibatkan penggunaan struktur data yang mendukung

penyimpanan data-thread dalam format yang terpisah. Sistem operasi atau lingkungan pengembangan menyediakan mekanisme untuk mendukung manajemen data khusus thread ini.

Contoh Penggunaan: Misalnya, dalam aplikasi pemesanan tiket pesawat, setiap permintaan dari klien untuk reservasi penerbangan dapat ditangani oleh thread yang berbeda. Setiap permintaan tersebut perlu diidentifikasi dengan ID unik untuk membedakan antara klien-klien yang berbeda. Untuk mengaitkan setiap thread dengan ID permintaan yang sedang ditangani, kita memerlukan data khusus thread.

DAFTAR PUSTAKA

Abas Ali Pangera, D.A. (2005) *Sistem Operasi*. Edited by Sigit Suryantoro. Yogyakarta: ANDI OFFSET.

Abraham Silberschatz, Peter Baer Galvin, G.G. (2019) *Operating System Concepts Ninth Edition*. 9th edn. New York: John Wiley and Sons.

Khurana, R. (2015) *Operating System Second Edition*. Second. Edited by ITLESL. Delhi: VIKAS PUBLISHING HOUSE PVT LTD.

Maya Utami Dewi (2021) *SISTEM OPERASI DAN UTILITY*. Edited by Agustinus Budi Santoso. Semarang: YPAT (Yayasan Prima Agus Teknik).

Nidhom, A.M. (2020) *Buku Ajar Sistem Operasi: Kernel System*. Edited by A.H. Nadhana. Malang: Ahli Media Press.

BAB 4
MANAJEMEN MEMORI DAN PENYIMPANAN

Oleh Herman Heriadi

4.1 Pendahuluan

Manajemen memori dan penyimpanan merupakan aspek krusial dalam mendesain dan operasi sistem komputer modern. Memori baik fisik maupun virtual adalah sumber daya yang harus dikelola dengan efisien untuk memastikan kinerja optimal dari perangkat lunak dan perangkat keras. Di sisi lain, penyimpanan adalah komponen penting yang memungkinkan penyimpanan jangka panjang data dan program, serta akses yang cepat dan andal terhadap informasi yang diperlukan oleh sistem dan pengguna.

Pada dasarnya, memori komputer terdiri dari memori fisik (RAM) dan memori sekunder (*Hard disk*, SSD) (Heryana dkk., 2023). Manajemen memori mengatur bagaimana memori fisik dialokasikan dan digunakan oleh berbagai proses yang berjalan pada sistem. Di sisi lain, memori virtual memungkinkan sistem operasi untuk mengatasi keterbatasan memori fisik dengan menggunakan sebagian penyimpanan sekunder untuk memperluas ruang memori yang tersedia.

Penyimpanan data melibatkan *file system* yang mengorganisasikan dan mengelola data pada perangkat penyimpanan seperti *hard disk* dan SSD. *File system* memungkinkan penyimpanan, pengambilan, dan pengelolaan data secara efisien dan aman (Sasra & Hadi, 2024). Manajemen penyimpanan juga mencakup penjadwalan disk dan teknik optimasi untuk memastikan akses data yang cepat dan reliabel.

Manajemen memori dan penyimpanan tidak hanya berfokus pada efisiensi dan kinerja, tetapi juga mencakup aspek keamanan dan keandalan (Komalasari, 2023). Fragmentasi memori, misalnya, dapat mempengaruhi kinerja sistem, sementara kegagalan disk

dapat mengakibatkan kehilangan data yang signifikan. Oleh karena itu, strategi seperti *backup* dan enkripsi data menjadi bagian integral dari manajemen penyimpanan modern.

Tujuan dari bab ini adalah untuk memberikan pemahaman mendalam mengenai konsep, teknik, dan tantangan dalam manajemen memori dan penyimpanan. Dengan memahami prinsip dasar dan strategi yang digunakan dalam mengelola memori dan penyimpanan, pembaca diharapkan mampu mengaplikasikan pengetahuan ini untuk meningkatkan kinerja dan keandalan sistem komputer.

4.2 Prinsip Manajemen Memori Virtual dan Fisik

Memori dalam konteks komputasi terdiri dari memori fisik dan memori virtual. Memori fisik merujuk pada RAM (*Random Access Memory*) yang terbatas, sedangkan memori virtual adalah teknik yang memungkinkan sistem operasi untuk menggunakan disk penyimpanan sebagai perpanjangan dari memori utama, sehingga memungkinkan eksekusi program yang memerlukan lebih banyak memori daripada yang tersedia secara fisik.

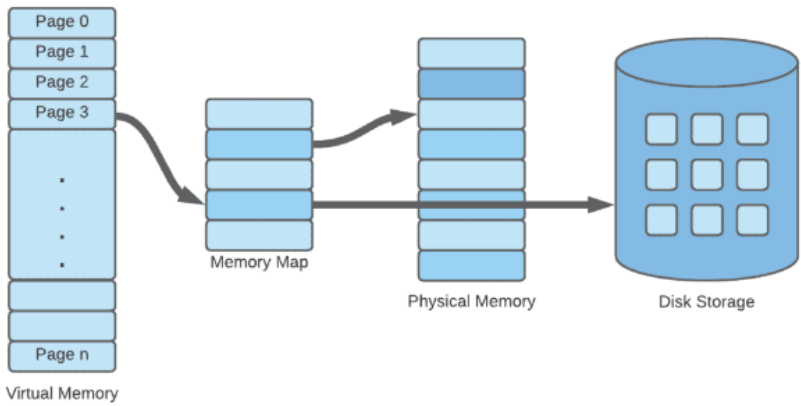
4.2.1 Memori Virtual

Memori virtual adalah teknik manajemen memori yang digunakan oleh sistem operasi untuk memberikan ilusi bahwa komputer memiliki lebih banyak memori fisik (RAM) daripada yang sebenarnya tersedia. Jarti & Hutabri (2022) mendefinisikan memori virtual sebagai sistem yang memungkinkan penggunaan sebagian dari memori sekunder, seperti *hard disk*, seolah-olah dijadikan sebagai memori internal utama. Dengan kata lain, sistem ini bekerja dengan memindahkan kode yang tidak diperlukan ke sebuah berkas di *hard drive* yang dikenal dengan istilah *page file*.

Teknik memori virtual memisahkan antara memori logis dan memori fisik (Josi, 2019). Memori logis terdiri dari seluruh halaman dari suatu program, dan tanpa memori virtual, memori logis akan langsung ditempatkan pada memori fisik. Dengan memori virtual, memori logis disimpan pada *secondary storage* (disk sekunder) dan hanya halaman yang diperlukan yang dipindahkan ke memori utama.

Teknik ini menyembunyikan detail fisik memori dari pengguna dengan mengubah menjadi memori sebagai alamat virtual yang berupa *byte* tanpa batas dan menempatkan beberapa bagian memori virtual ke dalam memori logis. Konsep ini pertama kali disampaikan oleh John Fotheringham pada tahun 1961 dengan menggunakan *dynamic storage allocation* pada sistem komputer Atlas di Universitas Manchester, dan istilah 'memori virtual' kemudian dipopulerkan oleh Peter J. Denning.

Memori virtual adalah strategi yang menciptakan gambaran memori yang sangat luas bagi pengguna, mengidealkan sumber daya terbatas agar dapat di akses oleh komputer (Tanduklangi & Amri, 2019). Ini memungkinkan untuk mengakses memori sekunder seolah-olah itu adalah bagian dari memori utama. Salah satu pendekatan untuk memperluas adalah dengan memindahkan penyimpanan RAM sementara yang tidak terpakai ke penyimpanan sekunder.



Gambar 4.1. Cara Kerja Memori Virtual
(Sumber : https://www.baeldung.com)

Pada dasarnya, memori virtual memungkinkan akses ke lebih banyak memori daripada yang sebenarnya tersedia pada sistem. Hal tersebut dicapai dengan memindahkan data dari RAM ke disk dengan bantuan sistem operasi dan perangkat lunak serta perangkat keras yang terinstall pada komputer.

4.2.2 Memori Fisik

Pada awalnya, komputer menggunakan jalur penundaan, tabung Williams-Kilburn, atau drum magnetis yang berputar sebagai penyimpanan utama. Pada tahun 1950-an, sebagian besar memori inti magnetik digantikan oleh teknologi yang lebih handal. Hingga tahun 1970-an, memori inti tetap dominan sampai memori semikonduktor menjadi lebih ekonomis berkat kemajuan teknologi *Integrated Circuit* (IC).

Memori fisik merujuk pada semua jenis perangkat keras yang digunakan untuk menyimpan data secara permanen atau sementara. Menurut Huffman (2014), memori fisik adalah bentuk penyimpanan data yang sangat cepat namun mudah berubah. Contoh paling umum dari memori fisik ialah *Random Access Memory* (RAM).

RAM dikenal sebagai memori fisik atau memori utama yang biasanya terletak pada *motherboard* (Stallings, 2015). Memori fisik ini bersifat mudah menguap dan ukurannya terbatas karena *chip* RAM. Oleh karena itu, diperlukan aliran daya yang konsisten untuk penyimpanan data. Masalah dan gangguan daya juga dapat menghapus data dari memori fisik.

Memori fisik dapat di akses secara linier, yang berarti memori ini tersusun secara berurutan. Memori fisik juga menyimpan instruksi program untuk dijalankan dan merupakan satu-satunya jenis penyimpanan yang dapat di akses langsung oleh CPU.

Gambar 4.2. Memori Fisik
(Sumber : https://unsplash.com)

Mengakses data melalui memori fisik membutuhkan waktu lebih sedikit dibandingkan mengakses data dari *hard drive*. Setelah dijalankan, program akan kembali ke *hard drive*, sehingga aplikasi baru dapat menggunakan memori bebas untuk dijalankan. Dengan kata lain proses ini dikenal sebagai "processes" ketika program mulai dijalankan.

4.2.3 Perbedaan Memori Virtual dan Fisik

Dalam dunia komputasi, terdapat dua jenis memori yang memiliki peran penting namun berbeda dalam cara kerjanya, yaitu memori fisik dan memori virtual. Kedua jenis memori ini saling melengkapi untuk memastikan bahwa sistem komputer dapat beroperasi dengan efisien dan dapat menjalankan program dengan kebutuhan memori yang beragam.

Tabel 4.1. Perbedaan Memori Virtual dan Fisik

	Memori Virtual	**Memori Fisik**
Definisi	Teknik manajemen memori	RAM sebenarnya
Pendekatan	Menggunakan halaman	Menggunakan pertukaran
Aksesibilitas	Tidak dapat mengakses CPU secara langsung	Bisa mengakses CPU
Ukuran	Terbatas pada ukuran disk	Terbatas pada ukuran RAM
Kecepatan	Lebih lambat dibandingkan memori fisik	Lebih cepat dari memori virtual

Sumber: https://www.baeldung.com

Memori fisik dan memori virtual adalah dua komponen yang saling melengkapi dalam manajemen memori komputer. Memori fisik menyediakan akses cepat untuk data yang sedang diproses, sementara memori virtual memungkinkan sistem untuk menjalankan program yang memerlukan lebih banyak memori

daripada yang tersedia secara fisik. Dengan bekerja bersama, keduanya memastikan bahwa sistem komputer dapat beroperasi dengan efisien, stabil, dan mampu menangani berbagai kebutuhan memori yang kompleks.

4.2.4 Prinsip – Prinsip Manajemen Memori

Manajemen memori sangat penting dalam sistem operasi untuk memastikan alokasi, penggunaan, dan pelepasan sumber daya memori secara efisien. Prinsip manajemen memori dalam sistem operasi komputer melibatkan beberapa konsep utama untuk mengoptimalkan penggunaan memori secara efisien. Terdapat beberapa prinsip-prinsip utama dalam manajemen memori yang membentuk dasar bagi desain dan implementasi sistem manajemen memori yang efektif

1. Alokasi Memori

   Proses di mana sistem operasi mengalokasikan memori fisik kepada proses yang sedang berjalan bertujuan untuk menghindari tumpang tindih yang dapat menyebabkan konflik dalam penggunaan memori. Dalam manajemen memori, cara memori dialokasikan kepada proses yang berjalan dalam sistem perlu diperhatikan. Alokasi memori dapat dilakukan secara statis (pada saat kompilasi), dinamis (selama waktu *runtime*), atau kombinasi dari keduanya. Sebagai contoh, Unix menggunakan teknik pemetaan memori yang efisien dengan menggunakan algoritma pemetaan berbasis lokasi, seperti algoritma *first-fit* atau *best-fit*.

2. Proteksi Memori

   Prinsip ini memastikan bahwa setiap proses hanya dapat mengakses memori yang diizinkan, sehingga akses ke memori dibatasi agar tidak ada proses yang dapat mengganggu atau merusak memori dari proses lain atau sistem operasi itu sendiri. Sistem operasi menyediakan mekanisme proteksi, seperti pengaturan akses memori (*read*, *write*, *execute*) dan pengaturan batas memori yang sesuai, untuk mencegah proses yang tidak berwenang mengakses atau mengubah data dari proses lain di memori.

3. Pemulihan Memori

   Prinsip ini mencakup strategi untuk memulihkan memori yang tidak lagi digunakan oleh proses agar dapat digunakan kembali, termasuk dealokasi memori yang tidak lagi diperlukan seperti yang dilakukan dalam manajemen memori dinamis. Proses ini penting untuk mengoptimalkan penggunaan sumber daya memori dan mencegah kebocoran memori, sehingga sistem dapat berjalan lebih efisien dan stabil.

4. *Paging* dan *Swapping*

   Prinsip ini melibatkan teknik untuk mengatasi keterbatasan memori fisik dengan menggunakan memori virtual. Teknik ini digunakan untuk mengelola memori fisik dan virtual, seperti *paging* dan *swapping*. Dalam *swapping*, sistem operasi akan memindahkan seluruh proses dari memori utama ke penyimpanan sekunder, seperti *hard disk*, untuk membebaskan ruang di memori utama bagi proses lain yang sedang di eksekusi. Sementara itu, *paging* memecah memori virtual dan fisik menjadi blok-blok kecil yang disebut halaman, di mana memori fisik dibagi menjadi *frame* dan memori logis menjadi *page*. *Page table* memetakan alamat logis ke alamat fisik, dan jika halaman tidak ada di memori fisik maka akan terjadi *page fault*, yang menyebabkan halaman tersebut dimuat dari penyimpanan sekunder (Silberschatz, dkk, 2018). Teknik ini memperluas memori fisik dengan menggunakan penyimpanan sekunder untuk meningkatkan efisiensi dan kinerja sistem.

5. Fragmentasi Memori

   Prinsip ini berkaitan dengan bagaimana alokasi dan dealokasi memori dapat menyebabkan fragmentasi, baik internal maupun eksternal. Manajemen fragmentasi memori melibatkan strategi untuk mengatasi atau meminimalkan dampak fragmentasi terhadap ketersediaan memori. Mencegah terjadinya fragmentasi memori yang dapat mengurangi efisiensi penggunaan memori adalah hal yang penting, sehingga sistem operasi menggunakan teknik defragmentasi atau pengaturan ulang memori untuk

mengelola ruang yang tersedia dan mengoptimalkan alokasi memori.

4.3 Strategi Alokasi Memori dan Manajemen *Swapping*

Manajemen memori memerlukan strategi yang efektif untuk alokasi dan dealokasi memori. Selain itu, mekanisme *swapping* digunakan untuk mengelola proses yang membutuhkan lebih banyak memori daripada yang tersedia. Dalam dunia komputasi modern, pengelolaan memori menjadi sangat penting untuk menjaga kinerja sistem operasi yang optimal. Strategi alokasi memori dan manajemen *swapping* adalah dua konsep inti dalam upaya ini. Strategi alokasi memori membahas cara sistem operasi mengelola dan menyediakan ruang memori kepada proses, sementara manajemen *swapping* merupakan teknik untuk mengatasi keterbatasan memori fisik dengan menggunakan penyimpanan sekunder.

4.3.1 Alokasi Memori

1. Alokasi Memori Berurutan (Kontigu)

   Alokasi memori berurutan (kontigu) adalah metode di mana setiap proses menempati satu blok tunggal lokasi memori yang berurutan (Wicaksana dan Rachman, 2018). Metode ini mengelola memori dengan menyediakan ruang yang diperlukan untuk program atau proses dalam urutan atau kesatuan yang berdekatan di dalam memori komputer. Dalam konteks ini, "berurutan" berarti blok-blok memori yang dialokasikan untuk sebuah program atau proses berdekatan secara fisik di dalam memori.

   Memori harus mampu melayani baik sistem operasi maupun proses pengguna, sehingga penting untuk mengalokasikan memori dengan efisien. Salah satu cara untuk mencapai efisiensi ini adalah dengan menggunakan alokasi memori berurutan. Dalam metode ini, alamat memori diberikan kepada proses secara berurutan dari kecil ke besar (Haryanto, 2012).

   Keuntungan dari alokasi memori berurutan mencakup efisiensi penyimpanan dan kecepatan akses. Efisiensi

penyimpanan tercapai karena proses dapat menyimpan data dalam urutan berurutan, memungkinkan penggunaan memori yang lebih optimal. Selain itu, kecepatan akses meningkat karena data disimpan dalam urutan yang berdekatan, mengurangi waktu yang dibutuhkan untuk mencari data.

Namun, alokasi memori berurutan juga memiliki kerugian, yaitu fragmentasi memori dan keterbatasan pertumbuhan. Fragmentasi memori terjadi ketika bagian-bagian kecil memori tidak dapat digunakan karena terpisah oleh bagian yang digunakan. Selain itu, kesulitan dalam mengelola memori muncul jika program atau proses membutuhkan lebih banyak memori daripada yang tersedia dalam urutan yang berdekatan, yang dapat mempercepat habisnya memori.

2. Alokasi Memori Tak Berurutan (Non-Kontigu)

Alokasi memori tak berurutan adalah metode pengelolaan memori di mana blok-blok memori yang diperlukan untuk program atau proses tidak disediakan dalam urutan atau kesatuan yang berdekatan di dalam memori komputer. Teknik ini sering digunakan pada sistem memori virtual, di mana alokasi halaman dilakukan secara global. Dalam konteks ini, blok-blok memori yang dialokasikan tidak berdekatan secara fisik di dalam memori.

Metode alokasi memori tak berurutan memiliki beberapa keuntungan, salah satunya adalah fleksibilitas. Teknik ini memungkinkan alokasi memori yang lebih fleksibel karena tidak terikat pada urutan berurutan. Selain itu, alokasi memori tak berurutan dapat mengurangi risiko fragmentasi memori karena blok-blok memori dapat ditempatkan di lokasi yang tersedia, meskipun tidak berdekatan.

Namun, alokasi memori tak berurutan juga memiliki beberapa kerugian. Salah satunya adalah penyimpanan yang tidak efisien. Memori mungkin tidak digunakan secara optimal karena blok-blok memori yang dialokasikan bisa tersebar di seluruh memori, meninggalkan celah yang tidak dapat digunakan. Selain itu, penyusutan kinerja juga bisa

terjadi karena akses ke data mungkin membutuhkan waktu lebih lama. Blok-blok memori yang tersebar memerlukan waktu tambahan untuk pencarian dan transfer data.

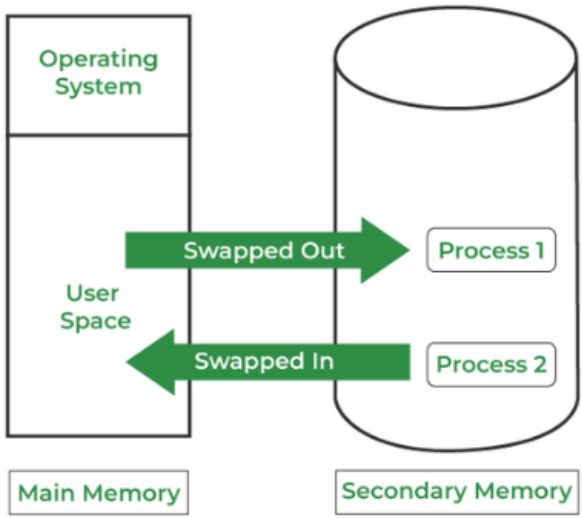
4.3.2 Manajemen *Swapping*

Manajemen *swapping* adalah teknik yang digunakan oleh sistem operasi untuk mengatasi keterbatasan memori fisik dengan memindahkan data antara memori fisik dan penyimpanan sekunder, seperti *hard disk* atau SSD. *Swapping* memungkinkan sistem operasi untuk menjalankan lebih banyak proses daripada yang dapat diakomodasi oleh memori fisik yang tersedia secara langsung.

Swapping adalah proses transfer data antara memori utama (RAM) dan penyimpanan sekunder (disk atau SSD) untuk mengelola kekurangan memori fisik (Tanenbaum & Bos, 2015). Tujuannya adalah untuk mengatasi keterbatasan memori fisik dengan memungkinkan sistem operasi untuk menjalankan lebih banyak proses atau mengakomodasi beban kerja yang lebih besar.

1. Proses *Swapping*

   Dalam konteks penukaran data, terjadi penyimpanan data yang tidak aktif atau jarang diakses dari proses yang sedang berjalan ke dalam penyimpanan sekunder seperti disk atau SSD. Ini adalah praktik yang umum dilakukan untuk mengelola penggunaan memori dalam sebuah sistem komputer. Ketika kapasitas memori fisik hampir mencapai batas maksimalnya, sistem mengambil langkah-langkah untuk memastikan keberlangsungan operasional dengan optimal.



Gambar 4.3. Proses *Swapping*
(Sumber : https://www.geeksforgeeks.org)

Salah satu langkah yang diambil adalah memindahkan data dari proses yang tidak aktif dari memori fisik ke penyimpanan sekunder. Proses ini, yang dikenal sebagai pemindahan data atau *swapping*, membebaskan ruang yang berharga dalam memori fisik untuk digunakan oleh proses baru yang mungkin memerlukan alokasi memori. Dengan demikian, penerapan penukaran data memungkinkan sistem untuk menjaga keseimbangan antara kebutuhan memori dari berbagai proses yang berjalan, memastikan bahwa sistem dapat beroperasi secara efisien dan responsif meskipun terdapat pembatasan pada kapasitas memori fisik yang tersedia.

2. Algoritma Swaping

Di dalam dunia sistem operasi, manajemen memori adalah salah satu aspek krusial untuk menjaga kinerja sistem agar tetap efisien dan responsif. Dua algoritma yang sering digunakan dalam konteks *swapping* memori adalah FIFO (*First In, First Out*) dan LRU (*Least Recently Used*). Mari kita

bahas kedua algoritma ini dari sudut pandang sistem operasi.

a. FIFO (*First In, First Out*)

   Algoritma FIFO bekerja dengan prinsip sederhana yaitu proses yang pertama kali dimasukkan ke dalam memori akan dikeluarkan terlebih dahulu saat terjadi kebutuhan untuk melakukan *swapping*. Ketika memori fisik tidak cukup untuk menampung semua proses yang aktif, sistem operasi akan memilih untuk mengeluarkan proses yang pertama kali dimasukkan ke dalam memori. Meskipun proses tersebut mungkin masih aktif, namun karena urutan masuknya lebih awal, proses ini dianggap lebih mudah untuk di-"swap out" agar memberikan ruang bagi proses baru yang membutuhkan memori lebih banyak.

b. LRU (*Least Recently Used*)

   Berbeda dengan FIFO, algoritma LRU mempertimbangkan pola akses terhadap memori secara lebih dinamis. Konsep utama dari LRU adalah bahwa proses yang paling lama tidak diakses atau jarang diakses akan menjadi kandidat utama untuk di-"swap out" saat memori fisik mulai mendekati kapasitas maksimumnya. Sistem operasi melacak waktu terakhir suatu proses diakses dan mengambil keputusan berdasarkan informasi ini. Proses yang jarang diakses cenderung memiliki prioritas lebih tinggi untuk di-"swap out" dibandingkan dengan proses yang sering diakses, karena memori dapat dialokasikan kembali kepada proses-proses yang lebih aktif.

3. Dampak dan Optimisasi *Swapping*

   Walaupun efektif dalam mengatasi keterbatasan memori fisik, *swapping* seringkali memiliki dampak negatif terhadap kinerja sistem. Proses ini dapat menyebabkan penurunan kinerja karena akses ke penyimpanan sekunder cenderung lebih lambat dibandingkan akses langsung ke memori fisik.

Dengan demikian, terjadi peningkatan waktu respons sistem dan menurunnya responsivitas aplikasi yang berjalan.

Namun, ada berbagai strategi optimisasi yang dapat diterapkan untuk mengurangi dampak negatif ini. Salah satunya adalah penggunaan algoritma penggantian halaman yang efisien, yang dapat meminimalkan jumlah *swap* yang diperlukan dan mengoptimalkan penggunaan memori. Selain itu, peningkatan jumlah memori fisik secara keseluruhan juga dapat membantu mengurangi kebutuhan akan *swapping*. Pendekatan lainnya adalah dengan melakukan pengaturan yang bijak terhadap kebutuhan aplikasi dan penggunaan memori, misalnya dengan mengalokasikan lebih banyak memori bagi aplikasi yang membutuhkan performa tinggi atau membatasi jumlah proses yang berjalan secara bersamaan untuk menghindari terjadinya *swapping* yang berlebihan. Sehingga, dampak negatif dari *swapping* dapat diminimalkan sehingga kinerja sistem dapat dipertahankan atau bahkan ditingkatkan.

4.4 Manajemen Penyimpanan

Penyimpanan merupakan elemen vital dalam setiap sistem komputer. Pada sub bab ini membahas berbagai aspek manajemen penyimpanan, mulai dari struktur *file system* hingga teknik disk management. Manajemen penyimpanan, termasuk *file system* dan *disk management*

Dalam ekosistem sistem komputer, manajemen penyimpanan memegang peran krusial dalam memastikan kinerja, keandalan, dan efisiensi sistem operasi (Silberschatz, dkk., 2018). Sebagai fondasi yang memungkinkan penyimpanan, organisasi, dan akses data, manajemen penyimpanan mencakup dua aspek utama: *file system*, yang bertanggung jawab untuk mengatur dan menyimpan file, dan disk management, yang mengelola fisik disk tempat data disimpan.

File system, sebagai titik awal pengorganisasian data, membentuk struktur hierarkis dari direktori dan sub-direktori yang memungkinkan pengguna dan aplikasi untuk menavigasi, menyimpan, dan mengambil data dengan mudah. Ini mencakup pengaturan data dalam file-file individu, serta implementasi

berbagai tipe *file system* yang dirancang untuk memenuhi kebutuhan khusus dari berbagai sistem operasi dan lingkungan penggunaan.

4.4.1 File system

File system atau sistem berkas adalah struktur hierarkis yang digunakan oleh sistem operasi untuk mengatur dan mengelola data pada perangkat penyimpanan seperti *hard drive*, SSD, atau media penyimpanan lainnya (Samsoni & Handayani, 2022). Ini mencakup cara data disimpan, diakses, diberi nama, dan diorganisir dalam hierarki direktori. Sistem berkas juga berfungsi untuk memastikan integritas data dan mengatur hak akses pengguna terhadap berkas dan direktori.

1. Fungsi *File system*
   a. Penyimpanan Data

      File system menyediakan mekanisme untuk menyimpan data secara terstruktur dalam format file, memungkinkan pengguna dan aplikasi untuk menyimpan informasi secara efisien.

   b. Organisasi Data

      File system mengorganisir data ke dalam file dan direktori, membentuk struktur hierarkis yang memudahkan pengelolaan dan penemuan informasi. Sehingga membantu pengguna untuk menavigasi dan mengelola data dengan lebih mudah.

   c. Akses Data

      File system memungkinkan sistem operasi, pengguna, dan aplikasi untuk mengakses data dengan cepat dan efisien dengan melibatkan operasi seperti membaca, menulis, dan menghapus file serta mengatur izin akses.

   d. Pengamanan Data

      File system menyediakan fitur keamanan seperti hak akses, enkripsi, dan integritas data untuk melindungi informasi dari akses yang tidak sah atau kerusakan dengan membantu menjaga kerahasiaan dan keutuhan data.

2. Jenis-jenis *File system*
   a. *File system* Berbasis Disk

      Contohnya FAT32, NTFS (Windows), ext4 (Linux), dan HFS+ atau APFS (macOS) merupkan *file system* yang digunakan untuk menyimpan data pada perangkat penyimpanan seperti *hard drive* atau SSD.
   b. *File system* Jaringan

      Digunakan untuk menyimpan dan mengakses file melalui jaringan komputer. Contoh *file system* jaringan termasuk NFS (*Network File system*) dan SMB (*Server Message Block*).
   c. *File system* Terdistribusi

      Menyimpan data di beberapa *node* dalam jaringan komputer yang terhubung. Contoh *file system* terdistribusi termasuk GFS (*Google File system*) dan HDFS (*Hadoop Distributed File system*).
3. Struktur *File system*

   Menurut Silberschatz, dkk. (2018), komponen-komponen utama dari struktur *file system* sebagai berikut.
   a. Konsep File

      File adalah unit penyimpanan logis dasar yang mewakili program, data, dan informasi lainnya.File memiliki berbagai atribut, seperti nama, pengenal, tipe, lokasi, ukuran, perlindungan, dan stempel waktu. Informasi tentang semua file disimpan dalam struktur direktori yang berada di perangkat yang sama dengan file.
   b. Operasi File

      Sistem operasi menyediakan panggilan sistem untuk melakukan operasi file dasar, seperti membuat, menulis, membaca, memposisikan ulang, menghapus, dan mengecilkan ukuran file. Untuk menghindari pencarian direktori yang terus-menerus, banyak sistem menggunakan tabel file terbuka untuk melacak informasi tentang file yang terbuka. Mekanisme penguncian file, baik wajib maupun nasihat, memungkinkan proses untuk menyinkronkan akses ke file yang dibagikan.

c. Tipe dan Struktur File

Sistem operasi mengenali berbagai tipe file, sering kali ditandai dengan ekstensi nama file, dan menyediakan dukungan untuk struktur file spesifik. Namun, mendukung beberapa tipe file dapat membuat sistem operasi besar dan kompleks, sehingga beberapa sistem memilih pendekatan yang lebih minimalis. Secara internal, file sistem harus menangani pemetaan catatan logis (misal, *byte*) ke blok fisik pada perangkat penyimpanan, yang dapat menyebabkan fragmentasi internal.

d. Metode Akses File

Dua metode akses utama adalah akses sekuensial, di mana file diproses berurutan, dan akses langsung (acak), di mana setiap blok dapat diakses secara langsung. Pilihan metode akses dapat berdampak signifikan pada kinerja dan kegunaan file sistem.

4.4.2 *Disk Management*

Disk management (pengelolaan disk) adalah proses mengelola dan mengoptimalkan penggunaan ruang penyimpanan pada perangkat keras seperti *hard drive* (HDD) atau *solid-state drive* (SSD). Dengan kata lain, *disk management* melibatkan berbagai tindakan, termasuk partisi disk, format disk, dan manajemen partisi yang ada untuk memastikan penggunaan ruang penyimpanan yang efisien dan efektif.

1. Fungsi *Disk management*

a. Partisi Disk

Partisi disk merupakan proses membagi ruang penyimpanan pada *hard drive* menjadi beberapa bagian logis yang terpisah, yang disebut partisi. Setiap partisi dapat berfungsi seperti *disk independen* dan dapat memiliki ukuran serta *file system* yang berbeda sesuai dengan kebutuhan pengguna, sehingga memungkinkan pengorganisasian data yang lebih efisien. Proses partisi disk biasanya melibatkan pembuatan tabel partisi yang mencatat informasi tentang setiap partisi, termasuk

ukuran, lokasi, dan jenis file system yang digunakan. Partisi disk sangat penting untuk mengatur ruang penyimpanan, meningkatkan kinerja, dan mempermudah manajemen data pada perangkat penyimpanan.

b. Format Disk

Format disk merupakan upaya untuk menghapus semua data dari disk dan mengatur ulang struktur *file system* sehingga disk siap digunakan untuk menyimpan data baru. Langkah ini diperlukan saat pertama kali mengatur disk atau jika ingin menghapus semua data dari disk dan membuat *file system* yang baru.

c. Manajemen Partisi

Manajemen partisi merupakan proses pengelolaan partisi pada perangkat penyimpanan seperti *hard drive* atau SSD, yang melibatkan kegiatan seperti pembuatan, penghapusan, pengubahan ukuran, dan pengaturan partisi untuk memenuhi kebutuhan pengguna. Hal ini memungkinkan pengguna untuk mengoptimalkan penggunaan ruang penyimpanan sesuai dengan kebutuhan mereka, memastikan data tersimpan dengan aman, dan meningkatkan kinerja sistem. Alat dan utilitas manajemen partisi sering kali mencakup fitur untuk memformat partisi, mengubah label partisi, dan memeriksa kesehatan partisi, sehingga memungkinkan pengguna untuk mengatur dan memelihara struktur penyimpanan dengan lebih efektif.

d. Pemantauan Kesehatan Disk

Pemantauan kesehatan disk merupakan proses yang bertujuan untuk memonitor kondisi fisik dan operasional dari media penyimpanan, seperti *hard drive* atau SSD. Tujuannya untuk mendeteksi potensi masalah atau kegagalan yang mungkin terjadi pada disk, seperti sektor yang rusak atau penurunan kinerja yang signifikan. Alat pemantauan kesehatan disk biasanya menyediakan informasi tentang suhu disk, jumlah sektor yang tidak stabil, tingkat keausan SSD (untuk SSD), dan parameter lain yang dapat memberikan petunjuk tentang kondisi

umum disk. Dengan memantau kesehatan disk secara teratur, pengguna dapat mengambil tindakan pencegahan seperti pencadangan data atau penggantian disk yang diperlukan untuk menghindari kehilangan data yang tidak diinginkan.

2. Teknik – teknik *Disk Management*
   a. Partisi Primer *vs.* Logis

      Partisi primer merupakan partisi utama pada disk yang dapat di-*boot* langsung oleh sistem operasi, sedangkan partisi logis merupakan partisi tambahan yang hanya bisa ada di dalam partisi ekstended. Partisi primer berdiri sendiri dan mampu berfungsi secara mandiri untuk *booting* komputer, sementara partisi logis terletak di bawah partisi ekstended dan tidak dapat diakses langsung oleh sistem operasi untuk *booting*. Penggunaan partisi logis berguna untuk mengatur ruang penyimpanan dalam partisi yang lebih terorganisir dan memungkinkan penggunaan lebih dari empat partisi pada satu *hard drive* dengan memanfaatkan struktur partisi ekstended.

   b. RAID (*Redundant Array of Independent Disks*)

      RAID (*Redundant Array of Independent Disks*) merupakan sebuah teknik yang mampu menggabungkan beberapa *hard drive* fisik menjadi satu entitas penyimpanan tunggal untuk meningkatkan kinerja, keandalan, atau keduanya. Terdapat beberapa tingkat RAID yang berbeda, seperti RAID 0, RAID 1, RAID 5, dan lain-lain, masing-masing dengan konfigurasi yang berbeda untuk tujuan tertentu. RAID 0 meningkatkan kinerja dengan menggabungkan beberapa *hard drive* ke dalam satu volume yang lebih besar tanpa redudansi data. RAID 1 menawarkan redudansi dengan menduplikasi data di dua atau lebih *hard drive* untuk melindungi terhadap kegagalan satu disk. RAID 5 menggunakan *striping* data dengan paritas untuk memberikan keseimbangan antara kinerja dan redudansi. RAID digunakan secara luas dalam server dan

sistem penyimpanan yang membutuhkan kinerja tinggi dan/atau perlindungan data yang handal.

c. *Disk Encryption*

Disk encryption merupakan proses mengamankan data yang tersimpan pada sebuah disk dengan mengubah teks biasa menjadi teks sandi (*ciphertext*) menggunakan algoritma enkripsi. Tujuan utama dari *disk encryption* adalah melindungi kerahasiaan data dari akses yang tidak sah atau pencurian fisik perangkat penyimpanan. Data yang dienkripsi tidak dapat dibaca tanpa kunci enkripsi yang sesuai, yang hanya diketahui oleh pihak yang sah. *Disk encryption* sering kali diterapkan pada level perangkat keras (seperti *hard drive* atau SSD) atau pada level *file system*, memastikan bahwa data tetap aman baik dalam keadaan aktif maupun diam.

d. *Backup* dan *Restore*

Backup dan *restore* merupakan proses penting dalam manajemen data yang melibatkan pencadangan (*backup*) dan pengembalian (*restore*) data untuk memastikan keamanan dan ketersediaan informasi yang penting. *Backup* merupakan salinan data yang disimpan secara terpisah dari data asli, yang dilakukan secara berkala untuk menghindari kehilangan data akibat kegagalan perangkat keras, serangan *malware*, atau kesalahan pengguna. Sedangkan, proses *restore* digunakan untuk mengembalikan data dari salinan *backup* tersebut ke sistem utama setelah terjadi kehilangan atau kerusakan data. Sehingga, proses ini merupakan bagian krusial dalam strategi pengelolaan data yang efektif, membantu mencegah kerugian data yang tidak diinginkan dan memulihkan operasional sistem dengan cepat dalam kasus bencana atau insiden keamanan.

DAFTAR PUSTAKA

Haryanto, E. V. 2012. *Sistem Operasi Konsep dan Teori*. Yogyakarta: Penerbit Andi.

Heryana, dkk. 2023. *Prinsip Sistem Operasi*. Banten: Sada Kurnia Pustaka.

Huffman, C. 2014. *Windows Performance Analysis Field Guide*. Elsevier.

Josi, A. 2019. *Sistem Operasi*. Bangka: Yayasan Kita Menulis.

Komalasari, dkk. 2023. *Pengantar Ilmu Komputer: Teori Komprehensif Perkembangan Ilmu Komputer Terkini*. Jambi: PT. Sonpedia Publishing Indonesia.

Samsoni & Handayani, Aprinia. 2022. *Organisasi Sistem Berkas*. Banten: Pascal Books.

Sasra, A. P., & Hadi, A. 2024. Perancangan Sistem Informasi Pendataan Ip Address Pada PT. KAI (Persero) Divre II Sumbar Berbasis Web Menggunakan Framework Laravel. *JEKIN-Jurnal Teknik Informatika, 4*(2), 49-56.

Silberschatz, A., Galvin, P. B. & Gagne, G. 2018. *Operating System Concepts*. 10th edn. Wiley.

Stallings, W. 2015. Computer *Organization and Architecture: Designing for Performance*. 10th edn. Pearson.

Jarti, N. & Hutabri, E. 2022. *Arsitektur Dan Organisasi Komputer*. Batam: Eureka Media Aksara.

Tanduklangi, A. & Amri, C. 2019. *Manajemen Sumber Daya Pembelajaran Bahasa Berbantuan Komputer: Computer Assisted Language Learning*. Deepublish.

Tanenbaum, A. S. & Bos, H. 2015. *Modern Operating Systems.* 4th edn. Pearson.

BAB 5
KEAMANAN DAN KINERJA

Oleh Purwita Sari

5.1 Konsep Dasar Keamanan Sistem Operasi

5.1.1 CIA Triad (*Confidentiality, Integrity, Availability*)

CIA *Triad* adalah model keamanan inti yang dirancang untuk memandu kebijakan keamanan informasi dalam suatu organisasi. Tiga komponen utamanya adalah:

1. *Confidentiality* (Kerahasiaan)

   Kerahasiaan mengacu pada perlindungan informasi dari akses yang tidak sah. Ini adalah prinsip fundamental dalam keamanan informasi yang memastikan bahwa data hanya dapat diakses oleh pihak yang berwenang.

   Implementasi:

   - Enkripsi: Menggunakan algoritma seperti AES atau RSA untuk mengenkripsi data saat disimpan (*at rest*) dan saat ditransmisikan (*in transit*).
   - Kontrol Akses: Menerapkan mekanisme autentikasi dan otorisasi untuk membatasi akses ke data sensitif.
   - Klasifikasi Data: Mengkategorikan data berdasarkan tingkat sensitivitasnya dan menerapkan kontrol yang sesuai.

   Tantangan:

   - Manajemen kunci yang kompleks dalam sistem berskala besar.
   - Keseimbangan antara kerahasiaan dan kemudahan akses untuk pengguna yang sah.

2. *Integrity* (Integritas)

   Integritas memastikan bahwa data tetap akurat, lengkap, dan tidak diubah oleh pihak yang tidak berwenang. Ini penting untuk menjaga keandalan dan konsistensi informasi dalam sistem operasi.

Implementasi:

- *Hashing*: Menggunakan fungsi hash kriptografis seperti SHA-256 untuk memverifikasi integritas data.
- *Digital Signatures*: Menggunakan tanda tangan digital untuk memastikan keaslian dan integritas pesan atau file.
- *Access Controls*: Membatasi siapa yang dapat memodifikasi data kritis.

Tantangan:

- Memastikan integritas dalam sistem terdistribusi.
- Mendeteksi dan mencegah manipulasi data yang halus.

3. *Availability* (Ketersediaan)

Ketersediaan memastikan bahwa informasi dan sumber daya sistem dapat diakses oleh pengguna yang berwenang saat diperlukan. Ini penting untuk menjaga produktivitas dan efisiensi operasional.

Implementasi:

- Redundansi: Menggunakan sistem cadangan dan load balancing.
- *Disaster Recovery*: Memiliki rencana dan infrastruktur untuk pemulihan cepat setelah gangguan.
- *Capacity Planning*: Memastikan sumber daya sistem memadai untuk menangani beban kerja.

Tantangan:

- Melindungi terhadap serangan *Denial of Service* (DoS).
- Menyeimbangkan ketersediaan dengan keamanan dan efisiensi biaya (Stallings and Brown, 2018).

5.1.2 Prinsip-prinsip Keamanan Dasar

1. *Least Privilege*

Prinsip *Least Privilege*, juga dikenal sebagai prinsip hak istimewa minimum, menyatakan bahwa setiap modul (seperti proses, pengguna, atau program) harus dapat mengakses hanya informasi dan sumber daya yang benar-benar diperlukan untuk menyelesaikan fungsinya (Saltzer and Schroeder, 1975). Ini adalah salah satu prinsip

keamanan yang paling fundamental dan penting dalam desain sistem komputer dan jaringan.

Konsep Utama:

a. Pembatasan Akses: Pengguna atau proses hanya diberikan akses ke sumber daya yang diperlukan untuk menyelesaikan tugas tertentu.
b. Granularitas: Hak akses diberikan dengan tingkat detail yang tinggi, memungkinkan kontrol yang tepat atas apa yang dapat diakses.
c. Prinsip *Need-to-Know*: Informasi hanya diberikan kepada mereka yang benar-benar membutuhkannya untuk melakukan pekerjaan mereka.

Implementasi:

a. *Role-Based Access Control* (RBAC): Hak akses dikaitkan dengan peran daripada individu, memungkinkan manajemen yang lebih efisien (Sandhu et al., 1996).
b. *Mandatory Access Control* (MAC): Sistem operasi menerapkan kebijakan keamanan yang ketat berdasarkan label keamanan.
c. *Discretionary Access Control* (DAC): Pemilik sumber daya menentukan hak akses untuk pengguna lain.
d. *Just-In-Time* (JIT) *Privileges*: Hak istimewa diberikan hanya saat diperlukan dan dicabut segera setelah selesai.
e. *Containerization*: Isolasi aplikasi untuk membatasi akses ke sumber daya sistem.

Manfaat:

a. Mengurangi Permukaan Serangan: Membatasi akses mengurangi jumlah vektor serangan potensial.
b. Mitigasi Dampak: Jika sebuah akun atau proses dikompromikan, dampaknya terbatas pada sumber daya yang dapat diaksesnya.
c. Audit yang Lebih Baik: Memudahkan pelacakan aktivitas dan identifikasi pelanggaran keamanan.

d. Stabilitas Sistem: Mencegah perubahan tidak sengaja pada komponen sistem kritis.

Tantangan:

a. *Overhead Administratif*: Manajemen hak akses yang granular dapat menjadi kompleks dan memakan waktu.
b. *Usability vs Security*: Keseimbangan antara keamanan dan produktivitas pengguna.
c. *Legacy Systems*: Sistem lama mungkin tidak dirancang dengan prinsip *least privilege*.
d. *Dynamic Environments*: Dalam lingkungan yang cepat berubah, mempertahankan prinsip *least privilege* bisa menjadi tantangan.

Contoh Implementasi:

a. Sudo di sistem *Unix-like*: Memungkinkan pengguna menjalankan program dengan hak keamanan pengguna lain, biasanya *superuser*.
b. *User Account Control* (UAC) di Windows: Meminta konfirmasi sebelum menjalankan tugas yang memerlukan hak administratif.
c. AppArmor dan SELinux: Mekanisme keamanan Linux yang membatasi kemampuan program berdasarkan profil yang ditentukan sebelumnya.

Best Practices:

a. Audit Berkala: Secara teratur meninjau dan memperbarui hak akses.
b. *Principle of Least Authority* (POLA): Memperluas least privilege ke desain sistem secara keseluruhan.
c. *Zero Trust Model*: Menerapkan verifikasi untuk setiap akses, tidak hanya pada batas jaringan.
d. *Privilege Separation*: Membagi aplikasi menjadi komponen dengan hak yang berbeda.

Perkembangan Terkini:

Dengan munculnya komputasi awan dan arsitektur mikroservis, prinsip least privilege semakin penting. Konsep seperti *Identity and Access Management* (IAM) di lingkungan *cloud* dan *Zero Trust Architecture* merupakan evolusi modern dari prinsip ini (Rose et al., 2020).

2. *Separation of Duties*

Prinsip ini melibatkan pembagian tugas kritis antara beberapa individu untuk mencegah penipuan atau kesalahan.

Implementasi:

- Pemisahan peran administrator sistem dan keamanan
- *Multi-person control* untuk operasi sensitif

Manfaat:

- Mengurangi risiko penyalahgunaan internal
- Meningkatkan akuntabilitas

Tantangan:

- Dapat meningkatkan kompleksitas operasional
- Memerlukan lebih banyak personel terlatih

3. *Defense in Depth*

Strategi keamanan yang menggunakan beberapa lapisan mekanisme keamanan untuk melindungi aset dan data. Konsep ini berasal dari strategi militer yang menerapkan pertahanan berlapis untuk memperlambat dan mencegah serangan (NIST, 2013).

Konsep Utama:

- Pertahanan Berlapis: Menggunakan berbagai mekanisme keamanan yang saling melengkapi.
- Redundansi: Memastikan bahwa kegagalan satu lapisan tidak mengkompromikan keseluruhan sistem.
- Diversifikasi: Menggunakan berbagai jenis kontrol keamanan untuk mengatasi berbagai jenis ancaman.

Implementasi:
Defense in Depth melibatkan beberapa lapisan keamanan:

a. Lapisan Fisik:
   - Kontrol akses fisik
   - Pengawasan video (CCTV)
   - Penjaga keamanan
b. Lapisan Jaringan:
   - *Firewalls*
   - *Intrusion Detection/Prevention Systems* (IDS/IPS)
   - *Virtual Private Networks* (VPNs)
   - *Network segmentation*
c. Lapisan Host:
   - Antivirus dan anti-malware
   - *Host-based firewalls*
   - *Endpoint Detection and Response* (EDR)
d. Lapisan Aplikasi:
   - *Web Application Firewalls* (WAF)
   - *Input validation*
   - *Encryption*
e. Lapisan Data:
   - *Enkripsi data at rest* dan *in transit*
   - *Data Loss Prevention* (DLP) *systems*
   - *Access controls*
f. Lapisan Administratif:
   - Kebijakan dan prosedur keamanan
   - Pelatihan kesadaran keamanan
   - *Manajemen patch* dan pembaruan (Whitman and Mattord, 2018).

Manfaat:

a. Perlindungan Komprehensif: Melindungi terhadap berbagai jenis ancaman.
b. Redundansi: Kegagalan satu lapisan tidak mengkompromikan seluruh sistem.
c. Deteksi Dini: Meningkatkan kemungkinan mendeteksi serangan sebelum mencapai aset kritis.

d. Waktu Respons: Memberikan lebih banyak waktu untuk merespons ancaman.

Tantangan:

a. Kompleksitas: Mengelola berbagai lapisan keamanan dapat menjadi rumit.
b. Biaya: Implementasi dan pemeliharaan beberapa lapisan keamanan bisa mahal.
c. Kinerja: Beberapa mekanisme keamanan dapat mempengaruhi kinerja sistem.
d. *False Positives*: Meningkatnya kompleksitas dapat menyebabkan lebih banyak *false positives*.

Best Practices:

a. *Risk Assessment*: Melakukan penilaian risiko menyeluruh untuk mengidentifikasi ancaman dan kerentanan.
b. *Layering Security Controls*: Menerapkan kontrol keamanan di berbagai tingkat infrastruktur IT.
c. *Regular Updates*: Memperbarui semua komponen keamanan secara teratur.
d. *Continuous Monitoring*: Memantau semua lapisan keamanan secara terus-menerus.
e. *Incident Response Planning*: Mengembangkan dan menguji rencana respons insiden.

Perkembangan Terkini:

Dengan munculnya *cloud computing* dan IoT, *Defense in Depth* telah berkembang untuk mencakup:

a. *Cloud Security Posture Management* (CSPM)
b. Container security
c. DevSecOps practices
d. *AI-driven security analytics* (Stallings and Brown, 2018).

Contoh Implementasi:

Sebuah organisasi menerapkan *Defense in Depth* dengan:

a. Menggunakan *firewall* jaringan dan *host-based*
b. Menerapkan enkripsi *end-to-end* untuk komunikasi

c. Menggunakan autentikasi multi-faktor
d. Menerapkan kebijakan *least privilege*
e. Melakukan pelatihan kesadaran keamanan rutin
f. Menggunakan sistem deteksi intrusi
g. Menerapkan segmentasi jaringan

4. *Fail-Safe Defaults*

Sistem harus defaultnya dalam keadaan aman, dan akses hanya diberikan setelah izin eksplisit.

Implementasi:

- *Default "deny all"* dalam konfigurasi *firewall*
- *Explicit allow* dalam kontrol akses *file system*

Manfaat:

- Mengurangi risiko konfigurasi yang tidak aman
- Memaksa pendekatan yang lebih hati-hati dalam pemberian akses

Tantangan:

- Dapat mempersulit konfigurasi awal sistem
- Berpotensi mengganggu penggunaan jika tidak dikelola dengan baik

5. *Complete Mediation*

Setiap akses ke setiap objek harus diperiksa untuk otorisasi.

Implementasi:

- *Continuous authentication*
- *Runtime access checks*

Manfaat:

- Mencegah *bypass* mekanisme keamanan
- Memastikan konsistensi dalam penegakan kebijakan keamanan

Tantangan:

- Dapat berdampak pada kinerja sistem
- Memerlukan desain yang cermat untuk mencegah *race conditions* (Saltzer and Schroeder, 1975).

5.1.3 Model Keamanan Sistem Operasi

1. *Bell-LaPadula Model*

   Model ini berfokus pada melindungi kerahasiaan informasi dan sering digunakan dalam sistem militer dan pemerintahan.

   Prinsip utama:
   - *No Read Up*: Subjek tidak dapat membaca objek pada level keamanan yang lebih tinggi.
   - *No Write Down*: Subjek tidak dapat menulis ke objek pada level keamanan yang lebih rendah.

   Implementasi:
   - *Multilevel security systems*
   - Klasifikasi data dan *clearance* pengguna

   Kelebihan:
   - Menyediakan perlindungan kuat terhadap kebocoran informasi
   - Cocok untuk lingkungan dengan hirarki keamanan yang jelas

   Keterbatasan:
   - Tidak memadai untuk melindungi integritas data
   - Dapat membatasi fleksibilitas dalam berbagi informasi

2. *Biba Model*

   Model Biba berfokus pada melindungi integritas data.

   Prinsip utama:
   - *No Read Down*: Subjek tidak dapat membaca objek pada level integritas yang lebih rendah.
   - *No Write Up*: Subjek tidak dapat menulis ke objek pada level integritas yang lebih tinggi.

   Implementasi:
   - Sistem yang memerlukan tingkat tinggi integritas data
   - Proteksi sistem file dan memori

   Kelebihan:
   - Menjaga integritas data dengan baik
   - Mencegah kontaminasi data dari sumber yang kurang terpercaya

Keterbatasan:

- Dapat membatasi aliran informasi yang diperlukan dalam beberapa skenario
- Tidak melindungi kerahasiaan

3. *Clark-Wilson Model*

   Model ini dirancang untuk sistem komersial dan berfokus pada integritas transaksi.

   Konsep utama:

   - *Well-formed transactions*
   - *Separation of duty*
   - *Auditing*

   Implementasi:

   - Sistem manajemen basis data
   - Sistem keuangan dan akuntansi

   Kelebihan:

   - Cocok untuk lingkungan bisnis
   - Menekankan pada akuntabilitas dan audit

   Keterbatasan:

   - Lebih kompleks untuk diimplementasikan dibandingkan model lainnya
   - Fokus pada integritas dapat mengabaikan aspek kerahasiaan

4. *Role-Based Access Control* (RBAC)

   RBAC adalah model yang mengaitkan hak akses dengan peran daripada dengan pengguna individual.

   Komponen utama:

   - *Users, Roles, Permissions, Sessions*

   Implementasi:

   - Sistem manajemen identitas dan akses modern
   - Sistem operasi *enterprise* dan aplikasi bisnis

   Kelebihan:

   - Memudahkan administrasi keamanan
   - Mendukung prinsip *least privilege*

Keterbatasan:

- Dapat menjadi kompleks dalam organisasi besar dengan banyak peran
- Memerlukan perencanaan dan pemeliharaan yang cermat (Sandhu et al., 1996).

5.1.4 Ancaman dan Serangan pada Sistem Operasi

1. Klasifikasi Ancaman

   Ancaman Internal vs Eksternal

   - Internal: dari dalam organisasi, sering melibatkan penyalahgunaan akses yang sah
   - Eksternal: dari luar organisasi, melibatkan upaya penetrasi dari jaringan eksternal

   Ancaman Pasif vs Aktif

   - Pasif: mencoba mendapatkan informasi tanpa mengganggu sistem (mis. penyadapan)
   - Aktif: mencoba memodifikasi sistem atau data (mis. malware, DoS)

2. Jenis-jenis Serangan
   - *Buffer Overflow:* memanfaatkan kerentanan dalam manajemen memori untuk mengeksekusi kode berbahaya
   - *Privilege Escalation*: meningkatkan hak akses melebihi yang seharusnya diberikan
   - Malware
     - Virus: menyebar dengan menginfeksi file lain
     - Worm: menyebar secara mandiri melalui jaringan
     - Trojan: menyamar sebagai *software* sah
   - *Denial of Service* (DoS): membanjiri sistem dengan permintaan untuk mengganggu layanan normal

3. Teknik Eksploitasi Sistem Operasi
   - *Zero-day vulnerabilities*: memanfaatkan kerentanan yang belum diketahui atau diperbaiki

- *Side-channel attacks*: menganalisis efek samping dari operasi sistem (mis. timing, power consumption) untuk mendapatkan informasi rahasia
- *Social engineering*: memanipulasi pengguna untuk melanggar prosedur keamanan

4. Evolusi Ancaman di Era Modern
    - *Ransomware*: malware yang mengenkripsi data korban dan meminta tebusan
    - *Targeted attacks*: serangan yang dirancang khusus untuk target tertentu (mis. APT - *Advanced Persistent Threat*)
    - *Supply chain attacks*: mengkompromikan perangkat lunak atau *hardware* dalam rantai pasokan
    - *Threats* dalam lingkungan cloud dan virtualisasi: serangan yang memanfaatkan karakteristik unik dari infrastruktur *cloud* (Szor, 2005; MITRE, 2021).

5.2 Kinerja Sistem Operasi: Konsep dan Metrik

5.2.1 Definisi Kinerja Sistem Operasi

Kinerja sistem operasi mengacu pada efisiensi dan efektivitas sistem dalam mengelola sumber daya komputasi dan memberikan layanan kepada pengguna dan aplikasi. Ini mencakup beberapa aspek kunci:

1. *Throughput*: Jumlah pekerjaan yang diselesaikan per unit waktu.
2. *Responsiveness*: Kecepatan sistem dalam menanggapi permintaan pengguna atau aplikasi.
3. *Utilization*: Tingkat penggunaan sumber daya sistem (CPU, memori, I/O).
4. *Scalability*: Kemampuan sistem untuk menangani peningkatan beban kerja.

Kinerja sistem operasi adalah hasil dari interaksi kompleks antara *hardware, software*, dan beban kerja (Stallings, 2018).

5.2.2 Metrik Pengukuran Kinerja

Beberapa metrik utama yang digunakan untuk mengukur kinerja sistem operasi meliputi:

1. CPU *Utilization*: Persentase waktu CPU yang digunakan untuk menjalankan tugas *non-idle.*
2. *Memory Usage*: Jumlah memori yang digunakan vs. yang tersedia.
3. *Disk I/O Rates*: Jumlah operasi baca/tulis disk per detik.
4. *Network Throughput*: Jumlah data yang ditransfer melalui jaringan per unit waktu.
5. *Response Time*: Waktu antara permintaan pengguna dan respons sistem.
6. *Latency*: Waktu yang dibutuhkan untuk menyelesaikan satu operasi.
7. MIPS (*Millions of Instructions Per Second*): Mengukur kecepatan prosesor.
8. FLOPS (*Floating-Point Operations Per Second*): Mengukur kinerja komputasi.

Pengukuran kinerja sering melibatkan *benchmarking*, yaitu menjalankan serangkaian tes standar untuk membandingkan kinerja sistem (Hennessy and Patterson, 2011).

5.2.1. Faktor-faktor yang Mempengaruhi Kinerja

Kinerja sistem operasi dipengaruhi oleh berbagai faktor, termasuk:

a. *Hardware Capabilities*:
   - Kecepatan CPU dan jumlah *core*
   - Kapasitas dan kecepatan memori
   - Kecepatan dan jenis penyimpanan (HDD vs SSD)
   - *Bandwidth* jaringan

b. Arsitektur Sistem Operasi:
   - Desain kernel (*monolithic vs microkernel*)
   - Algoritma penjadwalan
   - Manajemen memori dan *strategi caching*
   - Sistem file dan algoritma I/O

c. *Workload Characteristics*:

- Jenis aplikasi (CPU-*intensive*, I/O-*intensive*, *memory-intensive)*
- Tingkat konkurensi dan paralelisme
- Pola akses data

d. *Configuration and Tuning*:
- Parameter kernel
- *Buffer sizes*
- *Thread pools*
- *Resource allocation policies*

e. *Environmental Factors*:
- Temperatur dan kondisi fisik *hardware*
- *Network congestion*
- Interferensi elektromagnetik

f. *Software Optimization*:
- *Compiler optimizations*
- *Code efficiency*
- *Use of hardware acceleration* (e.g., GPU *computing*)

Memahami interaksi antara faktor-faktor ini penting untuk mengoptimalkan kinerja sistem operasi (Tanenbaum and Bos, 2015).

5.3 Optimisasi Kinerja Sistem Operasi

Optimasi kinerja sistem operasi merupakan aspek krusial untuk memastikan bahwa sistem berjalan secara efisien dan responsif. Ada beberapa area utama yang menjadi fokus dalam optimasi ini: manajemen proses dan penjadwalan, optimasi penggunaan memori, serta teknik *caching* dan *buffering* (Silberschatz, 2018).

5.3.1 Manajemen proses dan penjadwalan

Manajemen proses adalah aktivitas mengelola proses-proses yang berjalan di dalam sistem operasi. Setiap proses memerlukan CPU, memori, dan sumber daya lainnya. Manajemen proses mencakup pembuatan, penjadwalan, dan penghentian proses.

Penjadwalan proses menentukan urutan eksekusi proses untuk memaksimalkan efisiensi CPU. Beberapa algoritma penjadwalan meliputi:

1. *First-Come, First-Served* (FCFS): Proses yang datang pertama akan dijalankan terlebih dahulu. Sederhana tapi bisa menyebabkan masalah seperti *convoy effect.*
2. *Shortest Job Next* (SJN): Proses dengan waktu eksekusi terpendek akan dijalankan lebih dulu. Ini mengurangi waktu tunggu rata-rata, tapi memerlukan prediksi waktu eksekusi.
3. *Round Robin* (RR): Setiap proses mendapat jatah waktu eksekusi (quantum) secara bergiliran. Cocok untuk sistem *time-sharing.*
4. *Priority Scheduling*: Setiap proses diberi prioritas, dan proses dengan prioritas tertinggi dijalankan terlebih dahulu. Memerlukan mekanisme untuk mengatasi proses dengan prioritas rendah yang bisa terabaikan (*starvation*).

5.3.2 Optimasi penggunaan memori

Pengelolaan memori sangat penting untuk memastikan bahwa sistem tidak kehabisan memori dan dapat menjalankan proses secara efisien. Teknik yang sering digunakan meliputi:

1. *Paging*: Memori dibagi menjadi blok-blok kecil (*pages*), dan proses dapat dialokasikan ke halaman yang tidak berurutan. Ini mengurangi fragmentasi eksternal dan memungkinkan penggunaan memori lebih efisien.
2. *Segmentation*: Memori dibagi berdasarkan segmen logis seperti kode, data, dan stack. Setiap segmen dapat memiliki ukuran yang berbeda, yang membantu mengatur memori lebih fleksibel.
3. *Virtual Memory*: Memungkinkan sistem operasi untuk menggunakan *disk storage* sebagai perluasan RAM. Ini memungkinkan sistem menjalankan proses yang memerlukan lebih banyak memori dari pada yang tersedia secara fisik.

5.3.3 Teknik caching dan buffering

Caching dan *buffering* adalah teknik untuk meningkatkan kinerja dengan mengurangi waktu akses data:

1. *Caching*: Data yang sering diakses disimpan dalam *cache* (memori berkecepatan tinggi). CPU bisa mengakses data ini lebih cepat dibandingkan mengakses dari memori utama atau disk. Contohnya termasuk CPU *cache* dan *disk cache.*
2. *Buffering*: Data disimpan sementara dalam *buffer* selama proses transfer antara dua perangkat atau antara perangkat dan aplikasi. Buffering membantu menyamakan kecepatan antara perangkat yang berbeda dan mengurangi waktu tunggu (Bovet, 2005).

5.4 Alat-Alat yang Digunakan Untuk Analisis Kinerja Pada Keamanan Sistem Operasi

Analisis kinerja pada keamanan sistem operasi membutuhkan alat-alat yang dapat mengidentifikasi, mengukur, dan menganalisis potensi kerentanan serta performa keamanan. Berikut adalah beberapa alat penting yang sering digunakan untuk tujuan ini:

1. *Wireshark*

   Wireshark adalah salah satu alat paling populer untuk analisis jaringan. Alat ini memungkinkan pengguna untuk menangkap dan menganalisis lalu lintas jaringan secara mendetail. *Wireshark* berguna untuk mendeteksi aktivitas mencurigakan di jaringan, seperti serangan *man-in-the-middle* atau pencurian data.

   Fitur Utama:

   a. Mendukung berbagai protokol jaringan.
   b. Kemampuan untuk menangkap data secara *real-time.*
   c. Filter yang kuat untuk menyaring dan menganalisis paket data tertentu (Combs,2022).

2. *Snort*

   Snort adalah alat deteksi dan pencegahan intrusi jaringan yang sangat terkenal. *Snort* dapat menganalisis lalu lintas jaringan secara *real-time* dan mendeteksi berbagai jenis

serangan, termasuk *buffer overflow, stealth port scans,* dan OS *fingerprinting attempts.*

Fitur Utama:

a. Mendukung berbagai modus operasi, seperti *sniffer, packet logger*, dan *intrusion detection.*
b. Kemampuan untuk menulis aturan khusus untuk mendeteksi pola serangan.
c. Basis data yang luas tentang serangan dan kerentanan terbaru (Roesch, 1999).

3. Nmap

Nmap (*Network Mapper*) adalah alat *open-source* untuk eksplorasi jaringan dan audit keamanan. Nmap digunakan untuk memetakan jaringan, menemukan *host* dan layanan, serta mendeteksi kerentanan keamanan.

Fitur Utama:

a. Kemampuan untuk memindai port dan mendeteksi layanan yang berjalan di suatu host.
b. Identifikasi sistem operasi dan versi perangkat lunak yang berjalan.
c. Deteksi firewall dan filter jaringan (Lyon,2009).

4. OpenVAS

OpenVAS (*Open Vulnerability Assessment System*) adalah sistem untuk mengelola kerentanan dan alat penilaian yang menyediakan pemindaian kerentanan serta penilaian risiko.

Fitur Utama:

a. Pemindaian kerentanan yang komprehensif dengan berbagai plugin.
b. Manajemen kerentanan terpusat dengan laporan yang terperinci.
c. Integrasi dengan berbagai alat keamanan lainnya (Greenbone Networks, 2022).

5. Metasploit

   Metasploit adalah *framework* untuk pengujian penetrasi yang memungkinkan pengguna untuk mengidentifikasi, mengeksploitasi, dan memvalidasi kerentanan dalam sistem.
   Fitur Utama:
   a. Basis data eksploitasi yang luas.
   b. Kemampuan untuk melakukan serangan yang disimulasikan untuk menguji keamanan sistem.
   c. Antarmuka pengguna yang intuitif dan dukungan *scripting* untuk otomatisasi (Maynor, 2007).

6. Perf

   Perf adalah alat analisis kinerja yang digunakan di lingkungan Linux untuk memonitor kinerja CPU, memori, dan I/O. Perf dapat membantu mengidentifikasi *bottleneck* kinerja yang mungkin menjadi target serangan.
   Fitur Utama:
   a. Analisis mendalam tentang kinerja CPU dan memori.
   b. Kemampuan untuk merekam dan memutar ulang skenario kinerja.
   c. Dukungan untuk berbagai arsitektur hardware (Love, R., 2010).

DAFTAR PUSTAKA

Bovet, D. P., & Cesati, M. (2005). Understanding the Linux Kernel. 3rd Edition. O'Reilly Media.

Combs, G. (2022). Wireshark User's Guide. Wireshark Foundation.

Greenbone Networks. (2022). OpenVAS Administration Guide. Greenbone Networks GmbH.

Hennessy, J.L. and Patterson, D.A., 2011. Computer architecture: a quantitative approach. 5th ed. Waltham, MA: Morgan Kaufmann.

Love, R. (2010). Linux Kernel Development. Addison-Wesley Professional.

Lyon, G. F. (2009). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Com LLC.

Maynor, J. (2007). Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research. Syngress Publishing.

MITRE, 2021. MITRE ATT&CK®. [online] Available at: https://attack.mitre.org/ [Accessed 25 June 2024].

NIST, 2013. Security and Privacy Controls for Federal Information Systems and Organizations. NIST Special Publication 800-53, Revision 4.

Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. Proceedings of the 13th USENIX Conference on System Administration.

Rose, S., Borchert, O., Mitchell, S. and Connelly, S., 2020. Zero trust architecture. NIST Special Publication, 800(207), pp.1-50.

Saltzer, J.H. and Schroeder, M.D., 1975. The protection of information in computer systems. Proceedings of the IEEE, 63(9), pp.1278-1308.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E., 1996. Role-based access control models. Computer, 29(2), pp.38-47.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. 10th Edition. Wiley.
Stallings, W., 2018. Operating systems: internals and design principles. 9th ed. Harlow, UK: Pearson Education Limited.
Stallings, W. and Brown, L., 2018. Computer security: principles and practice. 4th ed. Harlow, UK: Pearson Education Limited.
Szor, P., 2005. The art of computer virus research and defense. Upper Saddle River, NJ: Addison-Wesley Professional.
Tanenbaum, A.S. and Bos, H., 2015. Modern operating systems. 4th ed. Upper Saddle River, NJ: Pearson Education.
Whitman, M.E. and Mattord, H.J., 2018. Principles of Information Security. 6th ed. Boston, MA: Cengage Learning.

BIODATA PENULIS

Dr. Teguh Wahyono
Dosen Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana

Penulis adalah dosen di Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana. Menyelesaikan pendidikan S1 Teknik Informatika Universitas Kristen Duta Wacana Yogyakarta tahun 1998, S2 Ilmu Komputer di Universitas Gadjah Mada Yogyakarta tahun 2011 dan S3 Ilmu Komputer di Universitas Bina Nusantara pada tahun 2021.

Bidang peminatannya dalam akademik adalah bidang *Data Science*, *Machine Learning* dan *Artificial Inteligence.* Di bidang tersebut telah mendapatkan sertifikasi *Deep Learning Specialist* dari *International Institute of Applied Informatics* (IIAI) Japan serta *Azure AI Fundamental Certified* dari Microsoft. Disamping itu yang bersangkutan juga mendapatkan Sertifikat Kompentensi dalam Bidang *Software Development* dari Badan Nasional Sertifikasi Profesi (BNSP) Republik Indonesia. Saat ini penulis mengampu beberapa mata kuliah yang diantaranya adalah *Software Engineering*, *Artificial Intelligence*, *Web and mobile programming* dan Teknologi Bisnis Digital.

BIODATA PENULIS

Dr. Lucky Indra Kesuma, S.SI., M.Kom
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Sjakhyakirti Palembang

Penulis Lahir di Palembang, 25 September 1990. Menyelesaikan pendidikan sebagai Ahli Madya pada program studi komputerisasi Akuntansi pada tahun 2011 di Universitas Sriwijaya Melanjutkan Pendidikan sebagai sarjana S1 program studi Sistem Informasi lulus pada tahun 2014 di Universitas Sriwijaya, menempuh pendidikan kembali S2 Magister Komputer Program Studi Teknik Informatika menyelesaikan studi pada tahun 2017 di Universitas Binadarma, Pada tahun 2023 telah selesai menempuh pendidikan S3 Program Doktor program studi Ilmu Teknik (Informatika). Penulis adalah Dosen tetap pada program studi Sistem Informasi Fakultas Ilmu Komputer Universitas Sjakhyakirti Palembang.

BIODATA PENULIS

Lutfi Syafirullah, S.T., M.Kom.
Dosen Program Studi Teknik Informatika
Jurusan Komputer dan Bisnis Politeknik Negeri Cilacap

Penulis lahir di Cilacap tanggal 21 November 1984. Penulis adalah Dosen Tetap pada Program Studi Teknik Informatika, Jurusan Komputer dan Bisnis, Politeknik Negeri Cilacap. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Ilmu Komputer. Saat ini penulis aktif melaksanakan Tri Dharma (pengajaran, penelitian dan pengabdian kepada Masyarakat) sejak tahun 2016 di Politeknik Negeri Cilacap.

BIODATA PENULIS

Herman Heriadi, S.Kom., M.Kom.
Dosen Program Studi Teknik Informatika
Universitas Dipa Makassar

Penulis lahir di Takkalasi tanggal 16 April 1992. Penulis adalah dosen tetap pada Program Studi Teknik Informatika, Universitas Dipa Makassar. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Sistem Informasi. Penulis menekuni bidang Menulis.

Senior Programmer dengan 6+ tahun pengalaman meggunakan Java untuk mengembangkan, telah mendesain dan mengembangkan lebih dari 10 aplikasi untuk meningkatkan efisiensi dan data management untuk perusahaan.

BIODATA PENULIS

Purwita Sari, S.SI., M.Kom.
Dosen Program Studi Manajemen Informatika
Fakultas Ilmu Komputer Universitas Sriwijaya

Penulis lahir di Palembang tanggal 09 Juni 1992. Penulis adalah dosen tetap pada Program Studi Manajemen Informatika Fakultas Ilmu Komputer, Universitas Sriwijaya. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknik Informatika di Universitas Bina Darma. Penulis menekuni bidang riset tentang Sistem Informasi.