

### *Struktur*

### *Sistem*

### *Operasi*

Sistem operasi menyediakan lingkungan di mana program dieksekusi. Secara internal, sistem operasi sangat bervariasi dalam tata rias mereka, dimana mereka diorganisir di sepanjang banyak jalur yang berbeda. Desain dari sistem operasi baru adalah tugas utama. Yang terpenting tujuan utamanya adalah sistem didefinisikan dengan baik sebelum desain dimulai. Tujuan-tujuan ini membentuk dasar untuk pilihan di antara berbagai algoritma dan strategi.

Kita dapat melihat sistem operasi dari beberapa titik yang menguntungkan. Satu pandangan berfokus pada layanan yang disediakan oleh sistem; lainnya, di antarmuka itu tersedia untuk pengguna dan programmer; ketiga, pada komponennya dan interkoneksi mereka. Dalam bab ini, kita akan mengeksplorasi ketiga aspek sistem operasi, menunjukkan sudut pandang pengguna, programmer, dan desainer sistem operasi. Kami mempertimbangkan layanan apa yang disediakan oleh sistem operasi, bagaimana mereka disediakan, bagaimana mereka dibebani, dan apa saja metodologi yang digunakan untuk merancang sistem semacam itu. Akhirnya, kami menjelaskan bagaimana sistem operasi dibuat dan bagaimana komputer memulai sistem operasinya.

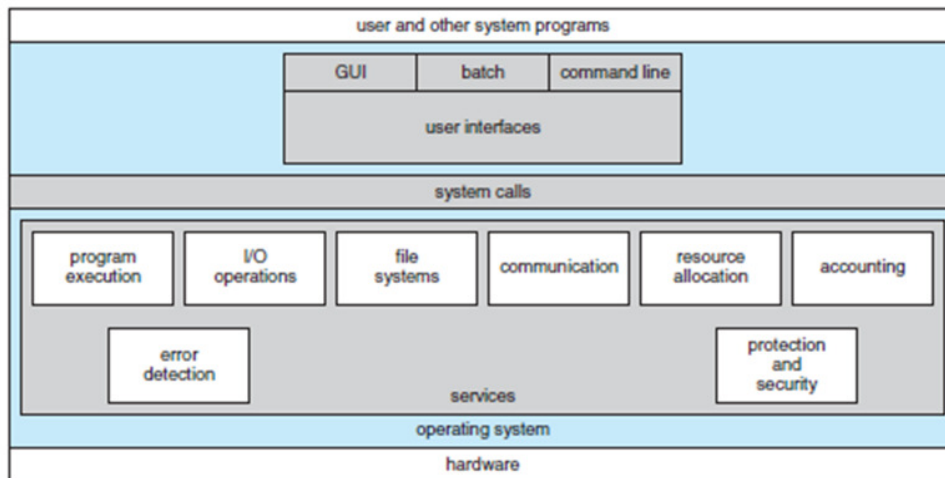
#### BAB TUJUAN

- Untuk menggambarkan layanan yang diberikan oleh sistem operasi kepada pengguna, proses, dan sistem lainnya.
- Untuk membahas berbagai cara penataan sistem operasi.
- Untuk menjelaskan bagaimana sistem operasi dipasang dan disesuaikan dan bagaimana mereka boot.

### 2.1 Layanan Sistem Operasi

Sistem operasi menyediakan lingkungan untuk pelaksanaan program. Sistem operasi juga menyediakan layanan tertentu untuk program dan kepada pengguna program-program tersebut. Layanan spesifik yang disediakan, tentu saja, berbeda dari satu sistem operasi ke yang lain, tetapi kita dapat mengidentifikasi kelas-kelas umum. Layanan sistem operasi ini disediakan untuk kenyamanan programmer, untuk membuat tugas pemrograman lebih mudah.

## Bab 2 Struktur Sistem Operasi



**Gambar 2.1** Tampilan layanan sistem operasi.

Gambar 2.1 menunjukkan satu pandangan dari berbagai layanan sistem operasi dan bagaimana mereka saling berhubungan.

Satu set layanan sistem operasi menyediakan fungsi-fungsi yang sangat membantu pengguna.

- **Antarmuka pengguna.** Hampir semua sistem operasi memiliki **antarmuka pengguna (UI)**. Antarmuka ini dapat mengambil beberapa bentuk. Salah satunya adalah **antarmuka baris perintah (CLI)**, yang menggunakan perintah teks dan metode untuk memasukkannya (misalnya, keyboard untuk mengetik perintah dalam format tertentu dengan pilihan yang spesifik). Lainnya adalah **antarmuka batch**, di mana perintah dan arahan untuk mengontrol perintah-perintah tersebut dimasukkan ke dalam file, dan file-file tersebut dieksekusi. Paling umum, **antarmuka pengguna grafis (GUI)** yang digunakan. Disini, antarmuka adalah sistem jendela dengan perangkat penunjuk untuk mengarahkan I/O, pilih dari menu, dan buat pilihan dan keyboard untuk memasukkan teks. Beberapa sistem menyediakan dua atau tiga variasi.
- **Eksekusi program.** Sistem harus dapat memuat program ke dalam memori dan menjalankan program itu. Program harus bisa mengakhiri eksekusinya, baik secara normal atau tidak normal (menunjukkan error).
- **I/O operasi.** Program yang sedang berjalan mungkin memerlukan I/O, yang mungkin melibatkan sebuah file atau perangkat I/O. Untuk perangkat tertentu, fungsi khusus mungkin diperlukan (seperti merekam ke drive CD atau DVD atau mengosongkan layar tampilan). Untuk efisiensi dan perlindungan, pengguna biasanya tidak dapat mengontrol perangkat I/O secara langsung. Oleh karena itu, sistem operasi harus menyediakan sarana untuk melakukan I/O.

## 2.1 Layanan Sistem Operasi

- **Manipulasi sistem file.** Sistem file sangat menarik. Sangat jelas, program perlu membaca dan menulis file dan direktori. Mereka juga perlu membuat dan menghapusnya dengan berdasarkan nama, pencarian file, dan daftar informasi file. Akhirnya, beberapa sistem operasi menyertakan izin manajemen untuk mengizinkan atau menolak akses ke file atau direktori berdasarkan file kepemilikan. Banyak sistem operasi menyediakan berbagai sistem file, kadang-kadang untuk memungkinkan pilihan pribadi dan kadang-kadang untuk memberikan fitur tertentu atau karakteristik kinerja.
- **Komunikasi.** Ada banyak keadaan di mana satu proses perlu bertukar informasi dengan proses lain. Komunikasi seperti itu dapat terjadi antara proses yang sedang dijalankan pada komputer yang sama atau antara proses yang mengeksekusi pada sistem komputer yang berbeda yang diikat bersama oleh jaringan komputer. Komunikasi dapat diimplementasikan melalui **memori bersama**, di mana dua proses atau lebih membaca dan menulis bagian dari memori bersama, atau **pesan yang lewat**, di mana paket informasi dalam format yang telah ditetapkan dipindahkan antar proses oleh sistem operasi.
- **Deteksi kesalahan.** Sistem operasi harus mendeteksi dan mengoreksi kesalahan terus-menerus. Kesalahan dapat terjadi di CPU dan perangkat keras memori (seperti kesalahan keseimbangan memori atau kegagalan daya), di perangkat I/O (seperti kesalahan keseimbangan pada disk, kegagalan koneksi pada jaringan, atau kurangnya kertas di printer), dan dalam program pengguna (seperti luapan aritmetika, upaya untuk mengakses lokasi memori ilegal, atau waktu penggunaan CPU yang terlalu banyak). Untuk setiap jenis kesalahan, sistem operasi harus mengambil tindakan yang tepat untuk memastikan komputasi yang benar dan konsisten. Terkadang, tidak ada pilihan selain dengan menghentikan sistem. Lain waktu, dengan mengakhiri proses yang menyebabkan kesalahan atau dengan mengembalikan kode kesalahan ke suatu proses untuk dideteksi oleh proses dan mungkin dapat diperbaiki.

Satu set fungsi sistem operasi ada bukan untuk membantu pengguna tetapi lebih untuk memastikan operasi yang efisien dari sistem itu sendiri. Sistem dengan beberapa pengguna dapat memperoleh efisiensi dengan berbagi sumber daya komputer di antara pengguna.

- **Alokasi sumber daya.** Ketika ada beberapa pengguna atau banyak pekerjaan berjalan pada saat yang sama, sumber daya harus dialokasikan untuk masing-masing pengguna. Sistem operasi mengelola berbagai jenis sumber daya. Sebagian (seperti siklus CPU, memori utama, dan penyimpanan file) mungkin memiliki kode alokasi khusus, sedangkan yang lain (seperti perangkat I/O) mungkin memiliki permintaan dan kode pelepasan

## Bab 2 Struktur Sistem Operasi

yang jauh lebih umum. Misalnya, dalam menentukan cara terbaik untuk menggunakan CPU, sistem operasi memiliki rutinitas penjadwalan CPU yang memperhitungkan kecepatan CPU, pekerjaan yang harus dijalankan, jumlah register yang tersedia, dan faktor lainnya. Mungkin juga ada rutinitas untuk mengalokasikan printer, drive penyimpanan USB, dan perangkat perifer lainnya.

- **Akuntansi.** Kita biasanya ingin melacak pengguna mana menggunakan seberapa banyak dan apa jenis sumber daya komputernya. Penyimpanan catatan ini dapat digunakan untuk akuntansi (sehingga pengguna dapat ditagih) atau hanya untuk statistik akumulasi penggunaan. Statistik penggunaan dapat menjadi alat yang berharga bagi peneliti yang menginginkan untuk mengkonfigurasi ulang sistem untuk meningkatkan layanan komputasi.
- **Perlindungan dan keamanan.** Pemilik informasi yang disimpan dalam multiuser atau sistem komputer berjaringan mungkin ingin mengontrol penggunaan informasi itu. Ketika beberapa proses terpisah dijalankan secara bersamaan, seharusnya tidak mungkin bagi satu proses dapat mengganggu proses yang lain atau operasi sistem itu sendiri. Perlindungan mencakup pemastian bahwa semua akses ke sistem sumber daya dikendalikan. Keamanan sistem dari pihak luar juga penting. Keamanan seperti itu dimulai dengan mengharuskan setiap pengguna untuk mengotentikasi diri sendiri ke sistem, biasanya dengan menggunakan kata sandi, untuk mendapatkan akses ke sumber daya sistem. Lebih luas lagi dapat juga digunakan untuk mempertahankan perangkat I/O eksternal, termasuk adapter jaringan, dari upaya akses yang tidak sah dan untuk merekam semua koneksi semacam itu untuk mendeteksi pembobolan. Jika suatu sistem harus dilindungi dan diamankan, tindakan pencegahan harus dilembagakan melalui itu semua. Rantai hanya sekuat ikatan terlemahnya.

### 2.2 Antarmuka Pengguna dan Sistem Operasi

Kami telah menyebutkan sebelumnya bahwa ada beberapa cara bagi pengguna untuk berinteraksi dengan sistem operasi. Di sini, kami membahas dua pendekatan mendasar. Satu menyediakan antarmuka baris perintah, atau **penerjemah perintah**, yang memungkinkan pengguna untuk langsung memasukkan perintah yang harus dilakukan oleh sistem operasi. Lainnya dengan memungkinkan pengguna untuk berinteraksi dengan sistem operasi melalui pengguna grafis antarmuka, atau GUI.

## 2.2 Antarmuka Pengguna dan Sistem Operasi

### 2.2.1 Penginterpretasi Perintah

Beberapa sistem operasi termasuk penerjemah perintah di kernel. Lainnya, seperti Windows dan UNIX, perlakukan penerjemah perintah sebagai program khusus yang berjalan saat pekerjaan dimulai atau ketika pengguna pertama kali masuk (dalam interaktif sistem). Pada sistem dengan beberapa penerjemah perintah untuk dipilih, interpreter dikenal sebagai **shell**. Misalnya, pada sistem UNIX dan Linux, seorang pengguna dapat memilih di antara beberapa shell yang berbeda, termasuk *Bourne Shell*, *C shell*, *Bourne-Again shell*, *Korn shell*, dan lainnya. Shell pihak ketiga dan shell gratis yang ditulis pengguna juga tersedia. Kebanyakan shell menyediakan fungsionalitas yang serupa, dan pilihan pengguna yang menggunakan shell umumnya didasarkan pada preferensi personal. Gambar 2.2 menunjukkan interpreter perintah shell Bourne yang digunakan di Solaris 10.

Fungsi utama dari penerjemah perintah adalah untuk mendapatkan dan mengeksekusi yang perintah berikutnya yang ditentukan pengguna. Banyak dari perintah yang diberikan pada tingkat ini memanipulasi file: buat, hapus, daftar, cetak, salin, eksekusi, dan sebagainya. MS-DOS dan UNIX shell beroperasi dengan cara ini. Perintah-perintah ini dapat diimplementasikan dalam dua cara umum.

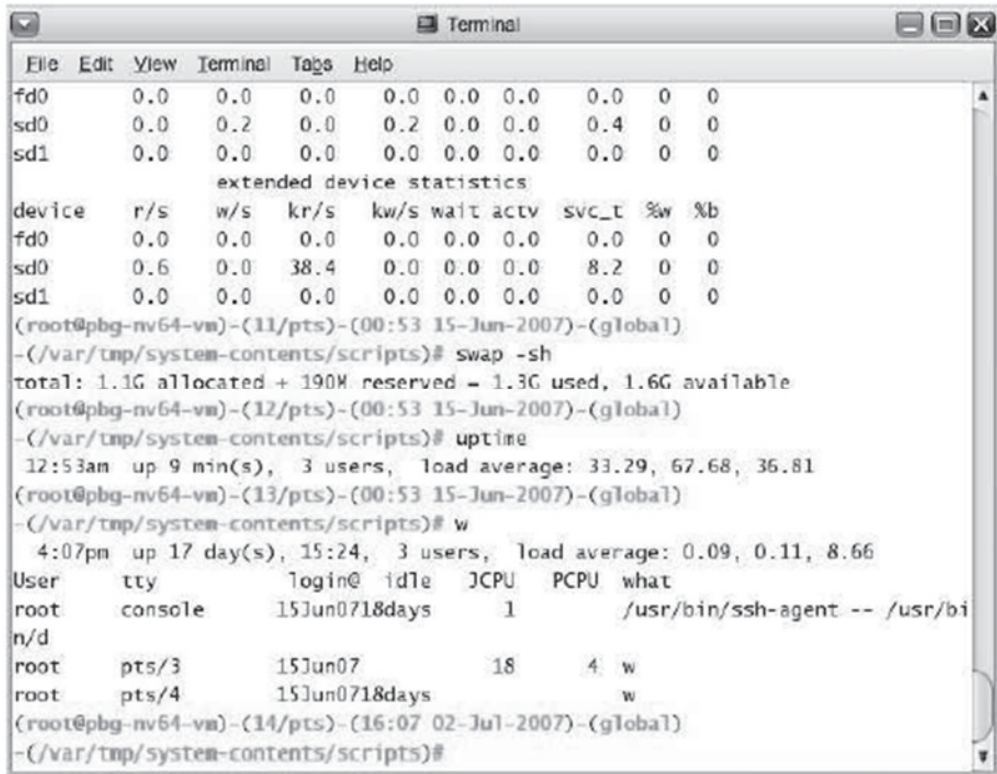
Dalam satu pendekatan, command interpreter itu sendiri berisi kode untuk jalankan perintah. Misalnya, perintah untuk menghapus file dapat menyebabkan perintah penerjemah untuk melompat ke bagian kode yang mengatur parameter dan membuat panggilan sistem yang sesuai. Dalam hal ini, jumlah perintah yang dapat diberikan menentukan ukuran interpreter perintah, karena setiap perintah membutuhkan kode implementasinya sendiri.

Pendekatan alternatif — digunakan oleh UNIX, di antara sistem operasi lainnya — Menyediakan sebagian besar perintah melalui program sistem. Dalam hal ini, penerjemah komando tidak memahami perintah dengan cara apa pun; melainkan hanya menggunakan perintah untuk mengidentifikasi file yang akan dimuat ke dalam memori dan dieksekusi. Jadi, perintah UNIX untuk menghapus file

```
rm file.txt
```

akan mencari file bernama `rm`, memuat file ke dalam memori, dan menjalankannya dengan parameter `file.txt`. Fungsi yang terkait dengan perintah `rm` akan didefinisikan sepenuhnya oleh kode dalam file `rm`. Dengan cara ini, programmer bisa menambahkan perintah baru ke sistem dengan mudah dengan membuat file baru dengan nama yang tepat. Program komando-penerjemah, yang bisa berukuran kecil, tidak perlu diubah untuk perintah baru yang akan ditambahkan.

## Bab 2 Struktur Sistem Operasi



```
File Edit View Terminal Tags Help
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.0 0.2 0.0 0.2 0.0 0.0 0.4 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
      extended device statistics
device r/s w/s kr/s kw/s wait actv svc_t %w %b
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.6 0.0 38.4 0.0 0.0 0.0 8.2 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User  tty      login@ idle  JCPU  PCPU  what
root  console  15Jun0718days  1     /usr/bin/ssh-agent -- /usr/bi
n/d
root  pts/3    15Jun07  18    4    w
root  pts/4    15Jun0718days  w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

Gambar 2.2 Interpretasi perintah shell Bourne di Solrais 10.

### 2.2.2 Antarmuka Pengguna Grafis

Strategi kedua untuk berinteraksi dengan sistem operasi adalah melalui antarmuka pengguna grafis yang ramah, atau GUI. Di sini, daripada memasukkan perintah langsung melalui antarmuka baris perintah, pengguna menggunakan jendela berbasis mouse dan sistem menu yang dicirikan oleh metafora **desktop**. Pengguna memindahkan mouse untuk memposisikan kursornya pada gambar, atau **ikon**, pada layar (desktop) yang mewakili program, file, direktori, dan fungsi sistem. Tergantung pada lokasi pointer mouse, mengklik tombol pada mouse dapat memanggil program, pilih file atau direktori — dikenal sebagai **folder** — atau tarik ke bawah menu yang berisi perintah.

Antarmuka pengguna grafis pertama kali muncul karena sebagian untuk penelitian yang terjadi pada awal 1970-an di fasilitas penelitian Xerox PARC. GUI pertama muncul pada komputer Xerox Alto pada tahun 1973. Namun, antarmuka grafis menjadi lebih banyak tersebar luas dengan munculnya komputer Apple Macintosh di tahun 1980-an. Antarmuka pengguna untuk sistem operasi Macintosh (Mac OS) telah mengalami berbagai perubahan selama bertahun-tahun, yang paling signifikan adalah adopsi antarmuka **Aqua** yang muncul dengan

Mac OS X. Versi pertama Microsoft dari Windows — Versi 1.0 — didasarkan pada penambahan antarmuka GUI ke Sistem operasi MS-DOS . Versi Windows yang lebih baru telah membuat perubahan kosmetik dalam tampilan GUI bersama dengan beberapa perangkat tambahan dalam fungsionalitas.

Karena mouse tidak praktis untuk sebagian besar sistem seluler, ponsel cerdas, dan komputer tablet genggam biasanya menggunakan antarmuka layar sentuh. Di sini, pengguna berinteraksi dengan membuat **gerakan** di layar sentuh — misalnya, menekan dan menggesekkan jari di layar. Gambar 2.3 mengilustrasikan layar sentuh dari Apple iPad. Padahal smartphone sebelumnya menggunakan keyboard fisik, sebagian besar smartphone sekarang mensimulasikan keyboard di layar sentuh.

Secara tradisional, sistem UNIX telah didominasi oleh antarmuka baris perintah. Berbagai antarmuka GUI tersedia, namun. Termasuk Common Desktop Environment (CDE) dan sistem X-Windows, yang umum pada versi komersial UNIX, seperti Solaris dan sistem AIX IBM. Selain itu, ada perkembangan yang signifikan dalam desain GUI dari berbagai proyek open source, seperti **K Desktop Environment** (atau **KDE** ) dan **GNOME** desktop oleh proyek GNU . Baik desktop KDE dan GNOME berjalan di Linux dan berbagai sistem UNIX dan tersedia di bawah lisensi open-source, yang berarti kode sumber mereka sudah tersedia untuk membaca dan untuk modifikasi di bawah ketentuan lisensi tertentu.



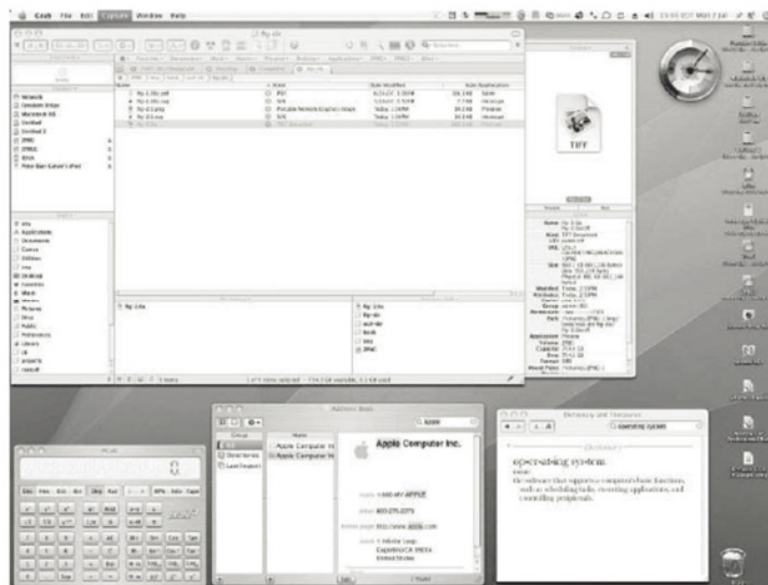
**Gambar 2.3** Layar sentuh iPad.

## 2.2 Antarmuka Pengguna dan Sistem Operasi

### 2.2.3 Pilihan Antarmuka

Pilihan apakah akan menggunakan antarmuka baris perintah atau GUI sebagian besar adalah preferensi pribadi. **Administrator sistem** yang mengelola komputer dan **pengguna daya** yang memiliki pengetahuan mendalam tentang sistem biasanya menggunakan antarmuka baris perintah. Bagi mereka, itu lebih efisien, memberi mereka akses yang lebih cepat ke aktivitas yang perlu mereka lakukan. Memang, pada beberapa sistem, subset fungsi sistem hanya tersedia melalui GUI, meninggalkan yang kurang umum tugas untuk mereka yang berpengalaman command-line. Selanjutnya, perintah-antarmuka baris biasanya membuat tugas yang berulang lebih mudah, sebagian karena mereka punya programabilitas mereka sendiri. Sebagai contoh, jika tugas yang sering membutuhkan satu set langkah-langkah baris perintah, langkah-langkah tersebut dapat direkam ke dalam file, dan file itu bisa dijalankan seperti program. Program ini tidak dikompilasi ke dalam kode yang dapat dieksekusi melainkan ditafsirkan oleh antarmuka baris perintah. **Skrip shell** ini adalah sangat umum pada sistem yang berorientasi pada command-line, seperti UNIX dan Linux.

Sebaliknya, sebagian besar pengguna Windows senang menggunakan Windows GUI lingkungan dan hampir tidak pernah menggunakan antarmuka shell MS-DOS. Beragam perubahan yang dialami oleh sistem operasi Macintosh memberikan studi yang bagus sebaliknya. Secara historis, Mac OS belum menyediakan antarmuka baris perintah, selalu mengharuskan penggunanya untuk berinteraksi dengan sistem operasi menggunakan GUI-nya. Namun, dengan rilis Mac OS X (yang sebagian diimplementasikan menggunakan UNIX kernel), sistem operasi sekarang menyediakan antarmuka Aqua dan antarmuka baris perintah. Gambar 2.4 adalah screenshot dari Mac OS X GUI.





## Gambar 2.4 Mac OS X GUI.

### Bab 2 Struktur Sistem Operasi

Antarmuka pengguna dapat bervariasi dari sistem ke sistem dan bahkan dari pengguna untuk pengguna dalam suatu sistem. Ini biasanya secara substansial dihapus dari yang sebenarnya struktur sistem. Desain antarmuka pengguna yang bermanfaat dan ramah karenanya bukan fungsi langsung dari sistem operasi. Dalam buku ini, kami berkonsentrasi pada masalah mendasar dalam menyediakan layanan yang memadai untuk program pengguna. Dari sudut pandang sistem operasi, kami tidak membedakan program pengguna dan program sistem.

### 2.3 Panggilan Sistem

**Panggilan sistem** menyediakan antarmuka ke layanan yang disediakan oleh operasi sistem. Panggilan ini umumnya tersedia sebagai rutinitas yang ditulis dalam C dan C++, meskipun tugas-tugas tingkat rendah tertentu (misalnya, tugas-tugas di mana perangkat keras harus diakses secara langsung) mungkin harus ditulis menggunakan bahasa assembly instruksi.

Sebelum kita membahas bagaimana sistem operasi membuat panggilan sistem tersedia, pertama mari kita gunakan contoh untuk mengilustrasikan bagaimana sistem panggilan digunakan: menulis program sederhana untuk membaca data dari satu file dan menyalinnya ke file lain. Masukan pertama yang dibutuhkan oleh program adalah nama dari dua file: file input dan file output. Nama-nama ini dapat ditentukan dalam banyak cara, tergantung pada desain sistem operasi. Salah satu pendekatan adalah untuk program untuk bertanya kepada pengguna untuk nama-nama. Dalam sistem interaktif, pendekatan ini akan membutuhkan urutan panggilan sistem, pertama-tama untuk menulis pesan yang mendorong di layar dan kemudian membaca dari keyboard karakter yang mendefinisikan dua file. Pada berbasis mouse dan sistem berbasis ikon, menu nama file biasanya ditampilkan di jendela. Pengguna kemudian dapat menggunakan mouse untuk memilih nama sumber, dan jendela dapat dibuka untuk nama tujuan yang akan ditentukan. Urutan ini membutuhkan banyak panggilan sistem I / O.

Setelah dua nama file diperoleh, program harus membuka masukan file dan buat file output. Setiap operasi ini membutuhkan yang lain panggilan sistem. Kemungkinan kondisi kesalahan untuk setiap operasi dapat membutuhkan tambahan panggilan sistem. Ketika program mencoba membuka file input, misalnya, mungkin menemukan bahwa tidak ada file dari nama itu atau bahwa file tersebut dilindungi terhadap akses. Dalam kasus ini, program harus mencetak pesan di konsol (yang lain urutan panggilan sistem) dan kemudian dihentikan secara tidak normal (panggilan sistem lain). Jika file input ada, maka kita harus membuat file output baru. Kita mungkin akan menemukan bahwa sudah ada file output dengan nama yang sama. Situasi ini dapat menyebabkan program untuk membatalkan (panggilan sistem), atau kita dapat menghapus file yang ada (panggilan sistem lain) dan membuat yang baru (panggilan sistem lain lagi). Pilihan lainnya, dalam sistem interaktif, adalah meminta pengguna (melalui urutan panggilan sistem ke output

## 2.3 Panggilan Sistem

pesan yang mendorong dan membaca tanggapan dari terminal) apakah akan mengganti file yang ada atau untuk membatalkan program.

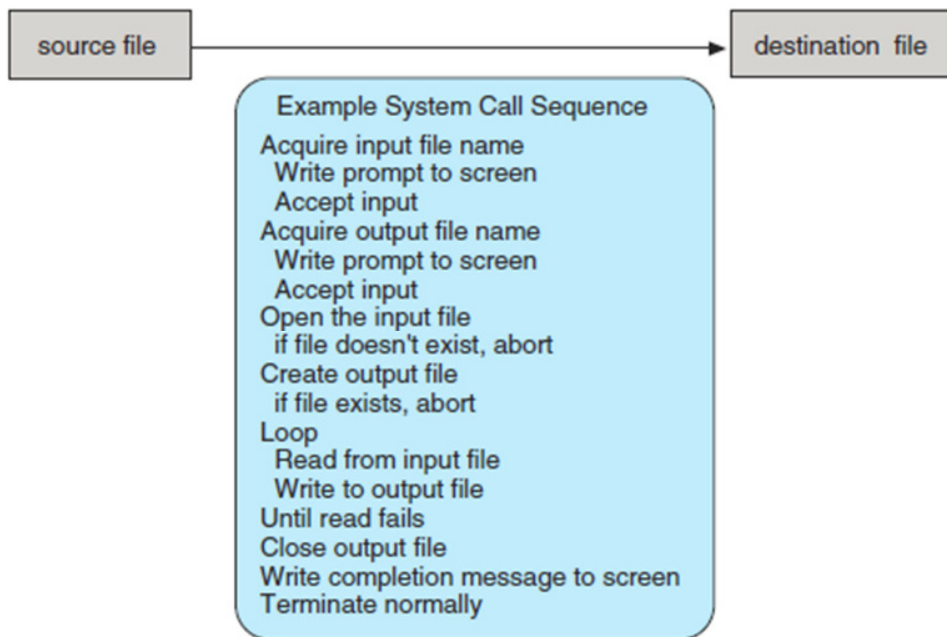
Ketika kedua file diatur, kami memasukkan loop yang membaca dari file input (panggilan sistem) dan menulis ke file output (panggilan sistem lain). Setiap membaca dan menulis harus dikembalikan informasi status mengenai berbagai kemungkinan kesalahan kondisi. Saat diinput, program dapat menemukan bahwa akhir dari file telah ada atau bahwa ada kegagalan perangkat keras dalam membaca (seperti kesalahan keseimbangan). Operasi tulis mungkin mengalami berbagai kesalahan, tergantung pada output perangkat (misalnya, tidak ada lagi ruang disk).

Akhirnya, setelah seluruh file disalin, program dapat menutup kedua file (panggilan sistem lain), tulis pesan ke konsol atau jendela (lebih banyak sistem panggilan), dan akhirnya berhenti secara normal (panggilan sistem terakhir). Panggilan sistem ini urutan ditunjukkan pada Gambar 2.5.

Seperti yang Anda lihat, bahkan program sederhana dapat memanfaatkan banyak sistem operasi. Seringkali, sistem melakukan ribuan panggilan sistem per detik. Kebanyakan programmer tidak pernah melihat level detail ini. Biasanya, pengembang aplikasi merancang program berdasarkan **aplikasi antarmuka pemrograman (Application Programming Interface / API)**. API menentukan satu set fungsi yang tersedia untuk programmer aplikasi, termasuk parameter yang ada lolos ke masing-masing fungsi dan mengembalikan nilai yang dapat diharapkan oleh programmer. Tiga API yang paling umum tersedia untuk pemrogram aplikasi Windows API untuk sistem Windows, API POSIX untuk sistem berbasis POSIX (yang mencakup hampir semua versi UNIX, Linux, dan Mac OS X), dan Java API untuk program yang berjalan di mesin virtual Java. Seorang programmer mengakses API melalui perpustakaan kode yang disediakan oleh sistem operasi. Dalam kasus UNIX dan Linux untuk program yang ditulis dalam bahasa C, perpustakaan disebut **libc**. Perhatikan bahwa — kecuali ditentukan — nama-nama panggilan sistem yang digunakan di seluruh teks ini adalah contoh umum. Setiap sistem operasi memiliki nama sendiri untuk setiap sistem memanggil.

Di belakang layar, fungsi-fungsi yang membentuk API biasanya memanggil panggilan sistem yang sebenarnya atas nama programmer aplikasi. Misalnya, Fungsi Windows `CreateProcess()` (yang tidak mengejutkan digunakan untuk membuat proses baru) sebenarnya memanggil `NT CreateProcess()` system call di Kernel Windows.

Mengapa programmer aplikasi lebih memilih pemrograman sesuai sebuah API daripada memanggil sistem panggilan yang sebenarnya? Ada beberapa alasan untuk melakukannya. Satu manfaat menyangkut portabilitas program.



**Gambar 2.5** Contoh bagaimana sistem panggilan digunakan.

## Jelaskan Fungsi Dari API!

Seorang programmer aplikasi yang merancang program menggunakan API dapat mengharapkan programnya untuk dikompilasi dan berjalan di sistem apa pun yang mendukung API yang sama (meskipun, dalam kenyataannya, arsitektur perbedaan sering membuat ini lebih sulit). Selanjutnya, panggilan sistem yang sebenarnya seringkali bisa lebih rinci dan sulit untuk dikerjakan daripada API yang tersedia untuk programmer aplikasi. Namun demikian, sering ada

korelasi kuat antara fungsi dalam API dan panggilan sistemnya yang saling terkait di dalam kernel. Bahkan, banyak API POSIX dan Windows yang serupa dengan panggilan sistem yang disediakan oleh UNIX, Linux, dan Windows. Untuk sebagian besar bahasa pemrograman, sistem pendukung run-time (satu set fungsi yang dibangun ke dalam pustaka yang disertakan dengan kompiler) menyediakan sebuah systemcall antarmuka yang berfungsi sebagai tautan ke panggilan sistem yang disediakan oleh sistem operasi. Antarmuka panggilan sistem memanggil fungsi panggilan di API dan memanggil panggilan sistem yang diperlukan di dalam sistem operasi. Khusus, nomor dikaitkan dengan setiap panggilan sistem, dan antarmuka panggilan sistem mempertahankan tabel yang diindeks sesuai dengan angka-angka ini. Antarmuka panggilan sistem kemudian memanggil panggilan sistem yang dimaksudkan dalam kernel sistem operasi dan mengembalikan status panggilan sistem dan nilai kembali apa pun. Penelepon tidak perlu tahu apa-apa tentang bagaimana panggilan sistem diimplementasikan atau apa yang dilakukannya selama eksekusi. Sebaliknya, pemanggil hanya perlu mematuhi API dan memahami apa yang akan dilakukan sistem operasi sebagai akibat dari eksekusi

panggilan sistem itu. Dengan demikian, sebagian besar detail antarmuka sistem operasi

disembunyikan dari programmer oleh API dan dikelola oleh run-time mendukung perpustakaan. Hubungan antara API, antarmuka panggilan sistem, dan sistem operasi ditunjukkan pada Gambar 2.6, yang menggambarkan bagaimana sistem operasi menangani aplikasi pengguna yang memohon panggilan sistem terbuka (). Panggilan sistem terjadi dalam berbagai cara, tergantung pada komputer yang digunakan. Seringkali, diperlukan lebih banyak informasi daripada sekadar identitas yang diinginkan panggilan sistem. Jenis dan jumlah informasi yang tepat dan bervariasi sesuai dengan sistem operasi tertentu dan panggilan. Misalnya, untuk mendapatkan masukan, kita mungkin membutuhkannya untuk menentukan file atau perangkat yang akan digunakan sebagai sumber, serta alamat dan panjang dari buffer memori di mana input harus dibaca. Tentu saja, perangkat atau file dan panjangnya mungkin tersirat dalam panggilan. Tiga metode umum digunakan untuk melewati parameter ke sistem operasi.

Pendekatan paling sederhana adalah melewati parameter dalam register. Dalam beberapa kasus, Namun, mungkin ada lebih banyak parameter daripada register. Dalam kasus ini,

parameter umumnya disimpan di blok, atau tabel, di memori, dan alamat blok dilewatkan sebagai parameter dalam register (Gambar 2.7). Ini adalah pendekatan yang diambil oleh Linux dan Solaris. Parameter juga bisa ditempatkan, atau mendorong, ke tumpukan oleh program dan memunculkan tumpukan oleh sistem operasi. Beberapa sistem operasi lebih memilih metode blok atau stack karena pendekatan tersebut tidak membatasi jumlah atau panjang parameter

dilewati.

## Jenis Panggilan Sistem

Panggilan sistem dapat dikelompokkan secara kasar ke dalam enam kategori utama: proses kontrol, manipulasi file, manipulasi perangkat, pemeliharaan informasi, komunikasi, dan perlindungan. Pada Bagian 2.4.1 hingga 2.4.6, kami secara singkat diskusikan jenis-jenis panggilan sistem yang mungkin disediakan oleh sistem operasi. Sebagian besar sistem ini mendukung panggilan, atau didukung oleh, konsep dan fungsi yang dibahas dalam bab-bab selanjutnya. Gambar 2.8 merangkum tipe-tipe sistem panggilan yang biasanya disediakan oleh sistem operasi. Seperti yang disebutkan, dalam teks ini, kita biasanya mengacu pada panggilan sistem dengan nama generik. Sepanjang teks, namun, kami memberikan yang sebenarnya dari panggilan sistem untuk sistem Windows, UNIX, dan Linux.

### 2.4.1 Kontrol Proses

Program yang berjalan harus dapat menghentikan eksekusinya secara normal (akhir ()) atau tidak normal (batalan ()). Jika panggilan sistem dibuat untuk mengakhiri saat ini menjalankan program secara tidak normal, atau jika program mengalami masalah dan menyebabkan jebakan kesalahan, dump memori kadang-kadang diambil dan kesalahan pesan yang dihasilkan. Dump ditulis ke disk dan dapat diperiksa oleh debugger — program sistem yang dirancang untuk membantu programmer dalam menemukan dan memperbaiki kesalahan, atau bug — untuk menentukan penyebab masalah. Dibawah baik keadaan normal atau abnormal, sistem operasi harus mentransfer kontrol ke penerjemah perintah pemohon. Kemudian penerjemah perintah membaca perintah selanjutnya. Dalam sistem interaktif, penerjemah perintah terus dengan perintah selanjutnya; diasumsikan bahwa pengguna akan

memberikan perintah yang tepat untuk menanggapi kesalahan apa pun. Dalam sistem GUI, sebuah jendela akan pop-up yang mungkin memperingatkan pengguna tentang kesalahan dan meminta panduan. Dalam sebuah sistem batch, interpreter perintah biasanya mengakhiri seluruh pekerjaan dan berlanjut dengan pekerjaan selanjutnya. Beberapa sistem memungkinkan pemulihan khusus dengan tindakan jika terjadi kesalahan. Jika program menemukan kesalahan dalam inputnya dan ingin mengakhiri secara tidak normal, mungkin juga ingin menentukan tingkat kesalahan. Kesalahan yang lebih parah dapat ditunjukkan oleh parameter kesalahan tingkat yang lebih tinggi.

Pengendalian proses

- mengakhiri, batalkan
- memuat, jalankan
- buat proses, hentikan proses
- dapatkan atribut proses, atur atribut proses
- menunggu waktu
- tunggu acara, acara sinyal
- mengalokasikan dan membebaskan memori
- Manajemen file

- buat file, hapus file
- buka, tutup
- membaca, menulis, memposisikan ulang
- dapatkan atribut file, atur atribut file
- Manajemen perangkat
  - meminta perangkat, lepaskan perangkat
  - membaca, menulis, memposisikan ulang
  - dapatkan atribut perangkat, atur atribut perangkat
  - pasang atau lepaskan perangkat secara logis
- Pemeliharaan informasi
  - dapatkan waktu atau tanggal, tetapkan waktu atau tanggal
  - dapatkan data sistem, atur data sistem
  - dapatkan atribut proses, file, atau perangkat
  - tetapkan proses, file, atau atribut perangkat
- Komunikasi
  - buat, hapus koneksi komunikasi
  - kirim, terima pesan
  - mentransfer informasi status
  - lampirkan atau lepaskan perangkat jarak jauh

Maka dimungkinkan untuk menggabungkan penghentian normal dan abnormal dengan mendefinisikan normal penghentian sebagai kesalahan di level 0. Perintah penerjemah atau yang berikut program dapat menggunakan tingkat kesalahan ini untuk menentukan tindakan selanjutnya secara otomatis.

Suatu proses atau pekerjaan yang mengeksekusi satu program mungkin ingin memuat () dan

execute () program lain. Fitur ini memungkinkan penerjemah perintah untuk jalankan program seperti yang diarahkan oleh, misalnya, perintah pengguna, klik mouse, atau perintah batch. Pertanyaan yang menarik adalah tempat untuk mengembalikan control ketika program yang dimuat berakhir. Pertanyaan ini terkait dengan apakah program yang ada hilang, disimpan, atau diizinkan untuk melanjutkan eksekusi secara bersamaan dengan program baru.

Jika kontrol kembali ke program yang ada saat program baru berakhir, kita harus menyimpan citra memori dari program yang ada; demikian, kami punyasecara efektif menciptakan mekanisme untuk satu program untuk memanggil program lain. Jika kedua program terus bersamaan, kami telah membuat pekerjaan atau

proses baru di-multiprogram. Seringkali, ada panggilan sistem khusus untuk tujuan ini (buat proses () atau kirimkan pekerjaan ()).

Jika kami membuat pekerjaan atau proses baru, atau mungkin bahkan satu set pekerjaan atau proses, kita harus dapat mengontrol eksekusinya. Kontrol ini membutuhkan kemampuan untuk menentukan dan mengatur ulang atribut pekerjaan atau proses, termasuk prioritas pekerjaan, waktu pelaksanaan maksimum yang diizinkan, dan seterusnya (dapatkan atribut proses () dan atur atribut proses ()). Kami mungkin juga ingin mengakhiri pekerjaan atau proses yang kami buat (menghentikan proses ()) jika kami menemukan bahwa itu tidak benar atau tidak lagi diperlukan. Setelah membuat pekerjaan atau proses baru, kita mungkin perlu menunggu mereka selesaikan eksekusinya. Kami mungkin ingin menunggu untuk waktu tertentu pass (waktu tunggu ()). Lebih mungkin, kami ingin menunggu acara tertentu terjadi (tunggu event ()). Pekerjaan atau proses seharusnya memberi isyarat ketika itu event telah terjadi (signal event ()). Cukup sering, dua atau lebih proses dapat berbagi data. Untuk memastikan integritas dari data yang dibagikan, sistem operasi sering menyediakan panggilan sistem yang memungkinkan sebuah proses untuk mengunci data bersama. Kemudian, tidak ada proses lain yang dapat mengakses data hingga kunci dilepaskan. Biasanya, panggilan sistem seperti itu termasuk memperoleh kunci () dan lepaskan kunci (). Sistem panggilan jenis ini, berurusan dengan koordinasi proses konkuren, didiskusikan dengan sangat rinci pada Bab 5.

Ada begitu banyak aspek dan variasi dalam proses dan kontrol pekerjaan itu kami selanjutnya menggunakan dua contoh — satu yang melibatkan sistem satu tugas dan sistem multitasking lainnya — untuk memperjelas konsep-konsep ini. Operasi MS-DOS sistem adalah contoh sistem satu tugas. Ini memiliki penerjemah perintah yang dipanggil saat komputer dimulai (Gambar 2.9 (a)). Karena MS-DOS adalah single-tasking, ia menggunakan metode sederhana untuk menjalankan program dan tidak membuatnya sebuah proses baru.

Ini memuat program ke dalam memori, menulis sebagian besar dari dirinya untuk memberikan program sebanyak mungkin memori (Gambar 2.9 (b)). Selanjutnya, ia mengatur penunjuk instruksi ke instruksi pertama dari program. Programnya kemudian berjalan, dan kesalahan bisa menyebabkan jebakan, atau program menjalankan panggilan system untuk mengakhiri. Dalam kedua kasus, kode kesalahan disimpan dalam memori sistem untuk nanti digunakan. Mengikuti tindakan ini, sebagian kecil dari penerjemah perintah yang tidak ditimpa melanjutkan eksekusi. Tugas pertamanya adalah mengisi ulang sisanya dari penerjemah perintah dari disk. Kemudian juru bahasa perintah membuatnya

kode kesalahan sebelumnya tersedia untuk pengguna atau ke program berikutnya. FreeBSD (berasal dari Berkeley UNIX) adalah contoh dari multitasking sistem. Ketika seorang pengguna log on ke sistem, shell pilihan pengguna dijalankan. Shell ini mirip dengan shell MS-DOS yang menerima perintah dan menjalankan program yang diminta pengguna. Namun, karena FreeBSD adalah a sistem multitasking, interpreter perintah dapat terus berjalan sementara

program lain dijalankan (Gambar 2.10). Untuk memulai proses baru, shell menjalankan panggilan sistem fork (). Kemudian, program yang dipilih dimuat ke dalam memori melalui panggilan sistem exec (), dan program dijalankan. Tergantung di jalan perintah itu dikeluarkan, shell kemudian menunggu prosesnya untuk menyelesaikan atau menjalankan proses "di latar belakang." Dalam kasus terakhir, shell segera minta perintah lain. Ketika sebuah proses sedang berjalan di latar belakang, tidak dapat menerima input langsung dari keyboard, karena shell menggunakan sumber daya ini. I / O karena itu dilakukan melalui

file atau melalui GUI antarmuka. Sementara itu, pengguna bebas meminta shell untuk menjalankan program lain, ke pantau kemajuan proses yang sedang berjalan, untuk mengubah prioritas program tersebut, dan seterusnya. Ketika proses selesai, ia akan mengeksekusi panggilan sistem keluar () ke mengakhiri, kembali ke proses memohon kode status 0 atau nol kode kesalahan. Status atau kode kesalahan ini kemudian tersedia untuk shell atau lainnya program. Proses dibahas dalam Bab 3 dengan contoh program menggunakan panggilan sistem fork () dan exec ().

### **2.4.2 Pengelolaan File**

Sistem file dibahas secara lebih rinci di Bab 11 dan 12. Kita bisa, Namun, identifikasi beberapa panggilan sistem umum yang berhubungan dengan file. Kita harus terlebih dahulu dapat membuat () dan menghapus () file. Entah panggilan sistem membutuhkan nama file dan mungkin beberapa atribut file. Sekali file dibuat, kita perlu membuka () dan menggunakannya. Kami juga dapat membaca (), tulis (), atau reposisi () (mundur atau lompat ke ujung file, misalnya). Akhirnya, kita perlu menutup () file, menunjukkan bahwa kita tidak lagi menggunakannya. Kami mungkin memerlukan set operasi yang sama ini untuk direktori jika kami memiliki struktur direktori untuk mengatur file dalam sistem file. Selain itu, untuk keduanya file atau direktori, kita harus dapat menentukan nilai-nilai dari berbagai atribut dan mungkin untuk mengatur ulang mereka jika perlu. Atribut file termasuk file nama, jenis file, kode perlindungan, informasi akuntansi, dan sebagainya. Paling sedikit dua panggilan sistem, dapatkan atribut file () dan atur atribut file (), adalah diperlukan untuk fungsi ini. Beberapa sistem operasi menyediakan lebih banyak panggilan, seperti panggilan untuk memindahkan file () dan menyalin (). Orang lain mungkin menyediakan API itu melakukan operasi tersebut menggunakan kode dan panggilan sistem lainnya, dan yang lainnya mungkin menyediakan program sistem untuk melakukan tugas-tugas itu. Jika program sistem dapat dipanggil oleh program lain, maka masing-masing dapat dianggap sebagai API oleh sistem lain program.

### **2.4.3 Manajemen Perangkat**

Suatu proses mungkin memerlukan beberapa sumber daya untuk dijalankan - memori utama, disk drive, akses ke file, dan sebagainya. Jika sumber daya tersedia, mereka dapat diberikan, dan kontrol dapat dikembalikan ke proses pengguna. Kalau tidak, prosesnya akan harus menunggu sampai sumber daya yang cukup tersedia. Berbagai sumber daya yang dikendalikan oleh sistem operasi dapat dipikirkan sebagai perangkat. Beberapa perangkat ini adalah perangkat fisik (misalnya, disk

drive), sementara yang lain dapat dianggap sebagai perangkat abstrak atau virtual (untuk contoh, file). Sistem dengan banyak pengguna mungkin meminta kami untuk meminta terlebih dahulu () perangkat, untuk memastikan penggunaan eksklusifnya. Setelah selesai dengan perangkat, kami lepaskan. Fungsi-fungsi ini mirip dengan sistem open () dan close () panggilan untuk file. Sistem operasi lain memungkinkan akses yang tidak dikelola ke perangkat.

menjalankan fork() system call. Kemudian, program yang dipilih dimuat ke memori melalui exec() system call, dan program ini dijalankan. Tergantung pada cara perintah itu dikeluarkan, shell tidak menunggu proses untuk menyelesaikan atau menjalankan proses "di latar belakang." Dalam kasus terakhir, shell segera permintaan perintah lain. Ketika sebuah proses sedang berjalan di latar belakang, itu tidak dapat menerima input langsung dari keyboard, karena shell menggunakan sumber daya ini. I/O oleh karena itu dilakukan melalui file atau melalui antarmuka GUI. Sementara itu, pengguna bebas untuk bertanya shell untuk menjalankan program lain, untuk memantau kemajuan proses yang sedang



berjalan, untuk mengubah program yang prioritas, dan sebagainya. Ketika proses selesai, itu mengeksekusi aplikasi yang `exit()` system call untuk menghentikan, kembali ke memohon proses kode status dari 0 atau nol kesalahan kode. Status ini atau kode kesalahan ini kemudian tersedia untuk shell atau program lain. Proses yang dibahas dalam Chapter 3 dengan contoh program menggunakan `fork()` dan `exec()` system calls.

## Manajemen File

file system ini dibahas secara lebih rinci dalam bab-Bab 11 dan 12. Namun, kita dapat mengidentifikasi beberapa umum system calls yang berhubungan dengan berkas. Pertama kita perlu untuk dapat membuat() dan hapus() file. Baik sistem call membutuhkan nama file dan mungkin beberapa file atribut. Setelah file dibuat, kita perlu `open()` dan untuk menggunakannya. Kita juga dapat membaca(), `write()`, atau reposisi() (mundur, atau melompat ke bagian akhir dari file tersebut, misalnya). Akhirnya, kita perlu untuk menutup() file, menunjukkan bahwa kita tidak lagi menggunakannya. Kita mungkin perlu ini sama set operasi untuk direktori jika kita memiliki sebuah struktur direktori untuk mengatur file dalam sistem file. Selain itu, untuk file atau direktori, kita harus mampu untuk menentukan nilai-nilai dari berbagai atribut dan mungkin untuk me-reset jika diperlukan. File attributes include nama file, jenis file, kode perlindungan, informasi akuntansi, dan sebagainya. Setidaknya dua sistem panggilan, mendapatkan atribut `file()` dan mengatur atribut `file()`, yang diperlukan untuk fungsi ini. Beberapa sistem operasi menyediakan lebih banyak panggilan seperti panggilan untuk memindahkan `file()` dan `copy()`. Orang lain mungkin menyediakan sebuah API yang melakukan operasi tersebut menggunakan kode-kode dan sistem panggilan, dan lain-lain mungkin memberikan sistem program untuk melakukan tugas-tugas. Jika sistem program yang memanggil mampu dengan program-program lainnya, pantai combers restoran program.

## DEVICE MANAGEMENT

Sebuah proses membutuhkan beberapa sumber daya untuk melaksanakan — memori utama, disk drive, akses ke file, dan sebagainya. Jika sumber daya yang tersedia, mereka dapat diberikan, dan kontrol dapat dikembalikan ke proses pengguna. Jika tidak, proses akan harus menunggu sampai sumber daya yang cukup tersedia.

Berbagai sumber daya yang dikendalikan oleh sistem operasi dapat dianggap sebagai perangkat. Beberapa dari perangkat ini adalah perangkat fisik (misalnya, disk drive), sementara yang lain dapat dianggap sebagai abstrak atau perangkat virtual (misalnya, file). Sistem dengan beberapa pengguna mungkin memerlukan kita untuk permintaan pertama() perangkat, untuk memastikan penggunaan eksklusif itu. Setelah kami selesai dengan perangkat ini, kita `release ()`. Fungsi-fungsi ini mirip dengan `open()` dan `close()` system calls untuk file. Sistem operasi lain yang memungkinkan tidak dikelola akses ke perangkat.

Bahaya maka potensi untuk perangkat perselisihan dan mungkin kebuntuan, yang dijelaskan dalam Bab 7.

Setelah perangkat telah diminta (dan dialokasikan kepada kita), kita bisa `read()`, `write()`, dan (mungkin) `reposition()` perangkat, sama seperti kita dapat dengan file. Bahkan, kesamaan antara perangkat I/O dan file yang begitu besar sehingga banyak sistem operasi, seperti UNIX, menggabungkan keduanya menjadi satu file gabungan – struktur perangkat. Dalam hal ini, seperangkat system call ini digunakan pada kedua file dan perangkat. Kadang-kadang, perangkat I/O yang diidentifikasi dengan nama file, direktori penempatan, atau atribut berkas.

User interface juga dapat membuat file dan perangkat yang tampak mirip, meskipun yang mendasari sistem panggilan yang berbeda. Ini adalah contoh lain dari banyak keputusan desain yang masuk ke dalam bangunan sistem operasi dan user interface.

## INFORMATION MAINTENANCE

Banyak sistem panggilan yang ada hanya untuk tujuan mentransfer informasi antara pengguna program dan sistem operasi. Misalnya, kebanyakan sistem memiliki sistem panggilan untuk kembali ke waktu saat `time()` dan `date()`. Lainnya sistem panggilan dapat kembali informasi tentang sistem, seperti jumlah pengguna saat ini, nomor versi sistem operasi, jumlah memori atau ruang disk, dan sebagainya.

Satu set sistem panggilan ini membantu dalam debugging program. Banyak sistem menyediakan system calls untuk `dump()` memori. Ketentuan ini berguna untuk debugging. Program jejak mencantumkan setiap sistem panggilan seperti itu dieksekusi. Bahkan mikroprosesor menyediakan modus CPU yang dikenal sebagai satu langkah, di mana perangkat dieksekusi oleh CPU setelah setiap instruksi. Perangkat biasanya tertangkap oleh sebuah debugger.

Banyak sistem operasi menyediakan waktu profil program untuk menunjukkan jumlah waktu bahwa program yang dijalankan di lokasi tertentu atau set lokasi. Waktu profil membutuhkan baik tracing fasilitas atau rutin interupsi timer. Pada setiap terjadinya interupsi timer, nilai program counter dicatat. Dengan cukup sering interupsi timer, gambar statistik dari waktu yang dihabiskan di berbagai bagian program dapat diperoleh.

Selain itu, sistem operasi menyimpan informasi tentang semua proses, dan system calls yang digunakan untuk mengakses informasi ini. Umumnya, panggilan juga digunakan untuk me-reset proses informasi (mendapatkan proses `getpgrp()` dan proses set `setpgrp()`). Di Bagian 3.1.3, kami membahas informasi apa yang biasanya disimpan.

Komunikasi

Ada dua model umum dari interprocess communication: pesan-lewat model dan bersama-model memori. Dalam pesan-lewat model, berkomunikasi proses pertukaran pesan dengan satu sama lain untuk mentransfer informasi. Pesan yang dapat dipertukarkan antara proses baik secara langsung atau secara tidak langsung melalui kotak pesan yang sama. Sebelum komunikasi dapat berlangsung, sambungan harus dibuka. Nama lain komunikator harus diketahui, baik itu proses lain pada sistem yang sama atau suatu proses pada komputer yang lain dihubungkan oleh jaringan komunikasi. Masing-masing komputer dalam jaringan memiliki host name dengan yang umum dikenal. Tuan rumah juga memiliki

network identifier, seperti alamat IP. Demikian pula, setiap proses memiliki proses nama, dan nama ini diterjemahkan ke dalam pengenalan dimana sistem operasi dapat merujuk kepada proses. Dapatkan `hostid()` dan mendapatkan `processid()` system calls melakukan terjemahan ini. Pengenal yang kemudian diteruskan ke tujuan umum `open()` dan `close()` panggilan yang diberikan oleh file sistem atau tertentu untuk membuka koneksi() dan menutup koneksi() system calls, tergantung pada sistem dan model komunikasi. Penerima proses biasanya harus memberikan izin untuk komunikasi berlangsung dengan menerima koneksi() call. Sebagian besar proses-proses yang akan menerima koneksi tujuan khusus daemon, yang sistem program yang disediakan untuk tujuan itu. Mereka mengeksekusi menunggu untuk koneksi() call dan terbangun saat sambungan dibuat. Sumber komunikasi, yang dikenal sebagai klien, dan menerima daemon, yang dikenal sebagai server, kemudian bertukar pesan dengan menggunakan pesan yang telah dibaca() dan menulis pesan() system calls. Penutupan koneksi() call mengakhiri komunikasi.

Di ruang-model memori, proses yang menggunakan memori bersama `create()` dan memori bersama `melampirkan()` system calls untuk membuat dan mendapatkan akses ke daerah memori yang dimiliki oleh proses lain. Ingat bahwa, biasanya, sistem operasi yang mencoba untuk mencegah satu proses dari mengakses proses lain di memori. Memori bersama yang membutuhkan dua atau lebih proses setuju untuk menghapus pembatasan ini. Mereka kemudian dapat bertukar informasi dengan membaca dan menulis data di tempat. Bentuk data ditentukan oleh proses dan tidak berada di bawah sistem operasi kontrol. Proses juga bertanggung jawab untuk memastikan bahwa mereka tidak menulis ke lokasi yang sama secara bersamaan. Mekanisme tersebut dibahas dalam Bab 5. Pada Bab 4, kita melihat variasi dari proses skema — benang — di mana memory ini dibagi oleh default.

Kedua model tersebut hanya dibahas secara umum dalam sistem operasi dan kebanyakan sistem menerapkan kedua. Pesan lewat berguna untuk bertukar jumlah yang lebih kecil dari data, karena tidak ada konflik yang perlu dihindari. Hal ini juga lebih mudah untuk menerapkan dari memori bersama untuk intercomputer komunikasi. Memori bersama memungkinkan kecepatan maksimum dan kenyamanan komunikasi, karena hal itu dapat dilakukan di memori kecepatan transfer ketika itu terjadi di dalam komputer. Masalah-masalah yang ada, namun, di bidang perlindungan dan sinkronisasi antara proses berbagi memori.

## Perlindungan

Perlindungan menyediakan mekanisme untuk mengontrol akses ke sumber daya yang disediakan oleh sistem komputer. Secara historis, perlindungan adalah perhatian hanya pada multiprogrammed sistem komputer dengan beberapa pengguna. Namun, dengan munculnya jaringan dan Internet, semua sistem komputer, dari server untuk mobile perangkat genggam, harus peduli dengan perlindungan.

Biasanya, system calls yang menyediakan perlindungan yang mencakup set (izin) dan mendapatkan izin(), yang memanipulasi izin pengaturan dari sumber daya seperti file dan disk. Memungkinkan pengguna() dan menyangkal user() system calls menentukan apakah pengguna tertentu bisa — atau tidak — akan diizinkan akses ke sumber daya tertentu.

Kami mencakup perlindungan di dalam Pasal 14 dan jauh lebih besar masalah keamanan dalam Bab 15.

## System programs

Aspek lain dari sistem modern adalah koleksi dari sistem program. Ingat Gambar 1.1, yang digambarkan logis komputer hirarki. Pada tingkat terendah adalah hardware. Berikutnya adalah sistem operasi, maka sistem program, dan akhirnya program aplikasi. Program sistem, juga dikenal sebagai sistem utilitas, menyediakan lingkungan yang nyaman untuk pengembangan program dan eksekusi. Beberapa dari mereka hanya user interface untuk sistem panggilan. Orang lain yang jauh lebih kompleks. Mereka dapat dibagi ke dalam kategori-kategori:

**Manajemen File.** Program-program ini membuat, menghapus, copy, rename, print, dump, daftar, dan umumnya memanipulasi file dan direktori.

**Informasi Status.** Beberapa program hanya meminta sistem untuk tanggal, waktu, jumlah, tersedia memori atau ruang disk, jumlah pengguna, atau status yang sama informasi. Orang lain yang lebih kompleks, menyediakan rinci kinerja, penebangan, dan informasi debugging. Biasanya, program-program ini format dan mencetak output ke terminal atau perangkat output lainnya atau file atau menampilkannya dalam jendela GUI. Beberapa sistem juga mendukung registry, yang digunakan untuk menyimpan dan mengambil informasi konfigurasi.

**Modifikasi File.** Beberapa teks editor yang tersedia untuk membuat dan memodifikasi isi berkas yang disimpan pada disk atau perangkat penyimpanan lainnya. Mungkin juga ada perintah khusus untuk mencari isi dari file atau melakukan transformasi dari teks.

**Bahasa pemrograman yang mendukung.** Penyusun, perakitan, debugger, dan interpreter untuk bahasa pemrograman umum (seperti C, C++, Java, dan PERL) yang sering ditawarkan dengan sistem operasi atau tersedia sebagai sebuah download yang terpisah.

**Program loading dan eksekusi.** Setelah program ini dirakit atau disusun, harus dimuat ke memori untuk dieksekusi. Sistem dapat memberikan absolute loader, relokasi loader, linkage editor, dan overlay loader. Debugging sistem untuk bahasa tingkat tinggi atau bahasa mesin yang diperlukan juga.

**Komunikasi.** Program-program ini menyediakan mekanisme untuk membuat koneksi virtual antar proses, user, dan sistem komputer. Mereka memungkinkan pengguna untuk mengirim pesan ke satu sama lain's

layar, untuk browsing halaman Web, mengirim e-mail, login jarak jauh, atau untuk mentransfer file dari satu mesin ke mesin lainnya.

Layanan latar belakang. Semua tujuan umum sistem memiliki metode untuk meluncurkan sistem tertentu-program proses pada saat boot. Beberapa dari proses ini menghentikan setelah menyelesaikan tugas-tugas mereka, sementara yang lain terus berjalan sampai sistem ini dihentikan. Terus-menerus menjalankan sistem-program proses yang dikenal sebagai pelayanan, subsistem, atau daemon. Salah satu contoh adalah jaringan daemon dibahas dalam Bagian 2.4.5. Dalam contoh ini, sistem yang membutuhkan layanan untuk mendengarkan untuk koneksi jaringan untuk menghubungkan permintaan tersebut ke proses yang benar. Contoh lain termasuk proses penjadwal yang memulai proses sesuai dengan jadwal yang telah ditentukan, sistem kesalahan pemantauan pelayanan, dan print server. Khas sistem memiliki puluhan

dari daemon. Selain itu, sistem operasi yang menjalankan kegiatan penting dalam konteks pengguna bukan dalam konteks kernel dapat menggunakan daemon untuk menjalankan kegiatan ini.

Bersama dengan sistem program, sebagian besar sistem operasi yang disertakan dengan program-program yang berguna dalam memecahkan masalah-masalah umum atau melakukan operasi yang paling umum. Seperti program-program aplikasi seperti browser Web, pengolah kata dan teks formatters, spreadsheet, database sistem, kompiler, merencanakan dan statistik-analisis paket, dan games.

Tampilan dari sistem operasi yang dilihat oleh sebagian besar pengguna didefinisikan oleh aplikasi dan sistem program, bukan oleh sistem yang sebenarnya panggilan. Mempertimbangkan pengguna PC. Ketika seorang pengguna komputer menjalankan sistem operasi Mac OS X, pengguna mungkin melihat GUI, menampilkan mouse-dan-antarmuka windows. Sebagai alternatif, atau bahkan di salah satu jendela, pengguna mungkin memiliki baris perintah UNIX shell. Keduanya menggunakan set yang sama dari sistem panggilan, tetapi sistem panggilan terlihat berbeda dan bertindak dalam cara yang berbeda. Lebih membingungkan pengguna melihat, mempertimbangkan pengguna dual-boot dari Mac OS X ke Windows. Sekarang pengguna yang sama pada hardware yang sama memiliki dua sepenuhnya antarmuka yang berbeda dan dua set aplikasi yang menggunakan sumber daya fisik. Pada hardware yang sama, maka, pengguna dapat terkena beberapa antarmuka pengguna secara berurutan atau bersamaan.

### Operasi-Desain dan Implementasi Sistem

Di bagian ini, kami membahas masalah yang kita hadapi dalam merancang dan mengimplementasikan sebuah sistem operasi. Ada, tentu saja, tidak ada solusi lengkap untuk masalah seperti itu, tapi ada pendekatan yang telah terbukti berhasil.

Masalah pertama dalam merancang suatu sistem adalah untuk menentukan tujuan dan spesifikasi. Pada tingkat tertinggi, desain sistem akan dipengaruhi oleh pilihan hardware dan jenis sistem: batch, waktu berbagi, single user, multiuser, didistribusikan, real time, atau tujuan umum.

Di luar ini tertinggi tingkat desain, persyaratan mungkin jauh lebih sulit untuk menentukan. Persyaratan bisa, bagaimanapun, dapat dibagi menjadi dua kelompok dasar: pengguna sasaran dan tujuan sistem.

Ingin pengguna tertentu yang jelas sifat dalam suatu sistem. Sistem harus mudah digunakan, mudah untuk dipelajari dan digunakan, dapat diandalkan, aman, dan cepat. Tentu saja, ini spesifikasi yang tidak terlalu

berguna dalam perancangan sistem, karena tidak ada perjanjian umum tentang bagaimana untuk mencapai mereka.

Satu set sama persyaratan dapat didefinisikan oleh orang-orang yang harus merancang, membuat, memelihara, dan mengoperasikan sistem. Sistem harus mudah untuk merancang, mengimplementasikan, dan memelihara; dan itu harus fleksibel, dapat diandalkan, bebas dari kesalahan, dan efisien. Sekali lagi, ini kebutuhan yang samar-samar dan dapat ditafsirkan dalam berbagai cara.

Ada, dalam jangka pendek, tidak ada solusi yang unik untuk masalah ini mendefinisikan persyaratan untuk sistem operasi. Berbagai macam sistem yang ada menunjukkan bahwa kebutuhan yang berbeda dapat mengakibatkan berbagai macam solusi untuk lingkungan yang berbeda. Misalnya, persyaratan untuk VxWorks, nyata-

saat sistem operasi untuk sistem embedded, harus telah secara substansial berbeda dari orang-orang untuk MVS, besar multiuser, multiaccess sistem operasi untuk IBM mainframe.

Menentukan dan merancang sebuah sistem operasi yang sangat kreatif tugas. Meskipun tidak ada buku yang dapat memberitahu anda bagaimana untuk melakukannya, prinsip-prinsip umum yang telah dikembangkan dalam bidang rekayasa perangkat lunak, dan sekarang kita beralih ke pembahasan beberapa dari prinsip-prinsip ini.

Salah satu prinsip penting adalah pemisahan kebijakan dari mekanisme. Mekanisme menentukan bagaimana melakukan sesuatu; kebijakan menentukan apa yang akan dilakukan. Misalnya, timer membangun (lihat Bagian 1.5.2) adalah sebuah mekanisme untuk memastikan CPU perlindungan, tetapi memutuskan berapa lama waktu yang harus ditetapkan untuk pengguna tertentu adalah keputusan kebijakan.

Pemisahan kebijakan dan mekanisme ini penting untuk fleksibilitas. Kebijakan mungkin berubah di tempat atau dari waktu ke waktu. Dalam kasus terburuk, setiap perubahan kebijakan akan membutuhkan perubahan dalam mekanisme yang mendasari. Umum mekanisme sensitif terhadap perubahan-perubahan kebijakan akan lebih diinginkan. Perubahan kebijakan akan membutuhkan redefinisi hanya parameter tertentu dari sistem. Sebagai contoh, pertimbangkan sebuah mekanisme untuk memberikan prioritas untuk jenis tertentu dari program-program yang lebih dari orang lain. Jika mekanisme ini benar dipisahkan dari kebijakan, hal ini dapat digunakan baik untuk mendukung keputusan kebijakan yang I/O yang intensif program harus memiliki prioritas di atas CPU-intensif yang atau untuk mendukung kebijakan yang berlawanan.

Mikrokernel berbasis sistem operasi (Bagian 2.7.3) mengambil pemisahan antara mekanisme dan kebijakan untuk satu ekstrim dengan menerapkan satu set dasar bangunan primitif. Blok ini hampir kebijakan bebas, yang memungkinkan lebih maju mekanisme dan kebijakan yang akan ditambahkan

melalui user-dibuat modul kernel atau pengguna program sendiri. Sebagai contoh, mempertimbangkan sejarah UNIX. Pada awalnya, itu memiliki waktu-berbagi scheduler. Dalam versi terbaru dari Solaris, penjadwalan dikendalikan oleh loadable tabel. Tergantung pada meja yang dimuat saat ini, sistem dapat pembagian waktu, batch processing, real time, adil, atau kombinasi. Membuat penjadwalan mekanisme tujuan umum memungkinkan beberapa perubahan kebijakan yang akan dibuat dengan satu beban-baru-tabel perintah. Pada ekstrem yang lain adalah sistem seperti Windows, di mana kedua mekanisme dan kebijakan yang dikodekan dalam sistem untuk menegakkan global terlihat dan merasa. Semua aplikasi memiliki antarmuka yang mirip, karena antarmuka itu sendiri dibangun dalam kernel dan sistem perpustakaan. Mac OS X sistem operasi yang memiliki fungsi serupa.

Keputusan kebijakan penting untuk semua alokasi sumber daya. Setiap kali anda perlu untuk memutuskan apakah atau tidak untuk mengalokasikan sumber daya, kebijakan, keputusan harus dibuat. Setiap kali pertanyaan adalah bagaimana daripada apa, itu adalah mekanisme yang harus ditentukan.

### 2.6.3 Pelaksanaan

Setelah sistem operasi ini dirancang, itu harus dilaksanakan. Karena sistem operasi adalah kumpulan dari banyak program, yang ditulis oleh banyak orang selama jangka waktu yang panjang, sulit untuk membuat pernyataan umum tentang bagaimana mereka diimplementasikan.

Awal sistem operasi yang ditulis dalam bahasa assembly. Sekarang, meskipun beberapa sistem operasi yang masih ditulis dalam bahasa assembly, sebagian besar ditulis dalam tingkat yang lebih tinggi bahasa seperti C atau yang lebih tinggi-tingkat bahasa seperti C++. Sebenarnya, sistem operasi dapat ditulis dalam lebih dari satu bahasa. Tingkat terendah dari kernel mungkin menjadi bahasa assembly. Tingkat yang lebih tinggi rutinitas mungkin di C, dan program sistem mungkin di C atau C++, di ditafsirkan bahasa scripting seperti PERL atau Python, atau shell script. Pada kenyataannya, mengingat distribusi Linux mungkin mencakup program-program yang ditulis dalam semua bahasa-bahasa tersebut.

Sistem pertama yang tidak ditulis dalam bahasa assembly adalah mungkin Master Control Program (MCP) untuk Burroughs komputer. MCP ditulis dalam varian ALGOL. MULTICS, yang dikembangkan di MIT, ditulis terutama dalam sistem bahasa pemrograman PL/1. Linux dan sistem operasi Windows kernel yang sebagian besar ditulis dalam C, meskipun ada beberapa bagian kecil dari kode assembly untuk driver perangkat dan untuk menyimpan dan memulihkan keadaan register.

Keuntungan menggunakan tingkat yang lebih tinggi bahasa, atau setidaknya sistem-implementasi bahasa, untuk menerapkan sistem operasi yang sama dengan yang diperoleh ketika bahasa digunakan untuk aplikasi program: kode dapat ditulis lebih cepat, lebih kompak, dan lebih mudah untuk memahami dan debug. Selain itu, perbaikan dalam compiler teknologi akan meningkatkan kode yang dihasilkan untuk

seluruh sistem operasi dengan penampilan sederhana. Akhirnya, sebuah sistem operasi yang jauh lebih mudah untuk port— pindah ke beberapa perangkat keras lainnya — jika itu yang tertulis di tingkat yang lebih tinggi itu. Misalnya, MS-DOS yang ditulis di Intel 8088 bahasa assembly. Akibatnya, hal ini berjalan secara native hanya pada keluarga Intel X86 Cpu. (Perhatikan bahwa meskipun MS-DOS yang berjalan secara native hanya pada Intel X86, emulator X86 set instruksi yang memungkinkan sistem operasi untuk berjalan pada Cpu lain — tapi lebih lambat, dan dengan penggunaan sumber daya lebih tinggi. Seperti yang telah disebutkan dalam Bab 1, emulator adalah program yang menduplikasi fungsi dari satu sistem pada sistem lain.) Sistem operasi Linux, sebaliknya, sebagian besar ditulis dalam C dan tersedia secara native pada jumlah Cpu yang berbeda, termasuk Intel X86, Oracle SPARC, dan IBM PowerPC.

Satu-satunya kemungkinan kerugian dari implementasi sebuah sistem operasi yang lebih tinggi-tingkat bahasa yang mengurangi kecepatan dan peningkatan persyaratan penyimpanan. Ini, bagaimanapun, adalah tidak lagi masalah besar dalam sistem hari ini. Meskipun seorang ahli perakitan-bahasa programmer dapat menghasilkan efisien kecil rutinitas, untuk program besar yang modern compiler dapat melakukan analisis kompleks dan menerapkan optimasi canggih yang menghasilkan kode yang sangat baik. Prosesor Modern memiliki mendalam pipelining dan beberapa unit fungsional yang dapat menangani rincian yang kompleks ketergantungan jauh lebih mudah daripada yang dapat pikiran manusia.

Seperti halnya di sistem lain, perbaikan kinerja utama di sistem operasi lebih mungkin untuk hasil yang lebih baik struktur data dan algoritma dari setiap majelis-kode bahasa. Selain itu, meskipun sistem operasi yang besar, hanya sejumlah kecil dari kode penting untuk kinerja tinggi; interrupt handler, I/O manager, manajer memori, dan CPU scheduler adalah mungkin yang paling penting rutinitas. Setelah sistem ini ditulis dan bekerja dengan benar, kemacetan rutinitas yang dapat diidentifikasi dan dapat diganti dengan majelis-bahasa setara.

## Operasi-Sistem Struktur

Sebuah sistem yang besar dan kompleks seperti sistem operasi modern harus direkayasa hati-hati jika itu adalah untuk berfungsi dengan baik dan dapat diubah dengan mudah. Pendekatan umum adalah untuk partisi task kedalam komponen-komponen kecil, atau modul, daripada memiliki satu sistem monolitik. Masing-masing modul-modul ini harus dapat didefinisikan dengan bagian dari sistem, dengan hati-hati didefinisikan input, output, dan fungsi. Kita telah membahas secara singkat di Bab 1 umum komponen-komponen dari sistem operasi. Di bagian ini, kita membahas bagaimana komponen-komponen ini saling berhubungan dan menyatu ke dalam kernel.

### 2.7.1 Struktur Sederhana

Banyak sistem operasi yang tidak memiliki didefinisikan dengan struktur. Sering, sistem tersebut dimulai sebagai kecil, sederhana, dan terbatas sistem dan kemudian tumbuh melampaui mereka asli lingkup. MS-DOS adalah contoh dari sistem tersebut. Ini pada awalnya dirancang dan dilaksanakan oleh beberapa orang yang tidak tahu bahwa itu akan menjadi begitu populer. Ini ditulis untuk memberikan fungsionalitas sedikit ruang, sehingga tidak hati-hati dibagi menjadi modul-modul. Gambar 2.11 menunjukkan struktur.



Dalam MS-DOS, antar muka dan tingkatan fungsionalitas tidak terpisah. Misalnya, program-program aplikasi yang mampu mengakses basic I/O rutinitas untuk menulis langsung pada layar dan hard disk. Seperti kebebasan daun MS-DOS rentan terhadap bandel (atau jahat) program, yang menyebabkan seluruh sistem crash ketika pengguna program gagal. Tentu saja, MS-DOS juga terbatas oleh hardware dari era-nya. Karena Intel 8088 yang ditulis tidak menyediakan dual mode dan tidak ada perlindungan keras, desainer dari MS-DOS tidak memiliki pilihan tetapi untuk meninggalkan dasar hardware yang dapat diakses.

Contoh lain dari terbatas penataan asli sistem operasi UNIX. Seperti MS-DOS, UNIX awalnya dibatasi oleh fungsi hardware. Ini terdiri dari dua bagian terpisah: kernel dan program sistem. Kernel ini selanjutnya dipisahkan ke rangkaian interface dan device driver, yang telah ditambah dan diperluas selama bertahun-tahun sebagai UNIX telah berkembang. Kita dapat melihat tradisional sistem operasi UNIX menjadi berlapis-lapis sampai batas tertentu, seperti yang ditunjukkan pada Gambar 2.12. Segala sesuatu di bawah system-call interface dan di atas perangkat keras fisik adalah kernel. Kernel menyediakan sistem file, penjadwalan CPU, manajemen memori, dan sistem operasi fungsi melalui system calls. Diambil singkatnya, itu adalah sejumlah besar fungsi yang akan digabungkan menjadi satu tingkat. Ini struktur monolitik adalah sulit untuk menerapkan dan memelihara. Ini memiliki kinerja yang berbeda keuntungan, namun ada sedikit overhead dalam sistem antarmuka panggilan atau komunikasi di dalam kernel. Kita masih melihat bukti ini sederhana, struktur monolitik di UNIX, Linux, dan sistem operasi Windows.

### 2.7.2 Pendekatan Berlapis

Dengan dukungan perangkat keras, sistem operasi dapat dipecah menjadi potongan-potongan yang lebih kecil dan lebih tepat daripada yang diperbolehkan oleh MS-DOS dan UNIX sistem. Sistem operasi kemudian dapat mempertahankan jauh lebih besar kontrol atas komputer dan aplikasi yang menggunakan komputer itu. Pelaksana memiliki lebih banyak kebebasan dalam mengubah cara kerja di dalam sistem dan dalam menciptakan modular sistem operasi. Di bawah pendekatan top-down, secara keseluruhan fungsi dan fitur yang ditentukan dan dipisahkan menjadi komponen-komponen. Menyembunyikan informasi ini juga penting, karena itu daun programmer bebas untuk menerapkan tingkat rendah rutinitas seperti yang mereka lihat cocok, asalkan antarmuka eksternal rutin tetap tidak berubah dan bahwa rutin melakukan diiklankan tugas.

Sebuah sistem dapat dibuat modular dalam banyak cara. Salah satu metode adalah pendekatan berlapis, di mana sistem operasi dibagi menjadi sejumlah lapisan (tingkat). Lapisan bawah (layer 0) adalah perangkat keras; tertinggi (lapisan N) adalah user interface. Ini layering struktur yang digambarkan pada Gambar 2.13.

Operasi-sistem lapisan adalah implementasi dari objek abstrak yang terdiri dari data dan operasi yang bisa memanipulasi data tersebut. Khusus sistem operasi layer — mengatakan, lapisan M — terdiri dari struktur data dan satu set rutinitas yang dapat dipanggil oleh tingkat yang lebih tinggi lapisan. Lapisan M, pada gilirannya, dapat menjalankan operasi pada tingkat yang lebih rendah lapisan.

Keuntungan utama dari pendekatan berlapis adalah kesederhanaan konstruksi dan debugging. Lapisan yang dipilih sehingga masing-masing menggunakan fungsi (operasi) dan jasa hanya lebih rendah-tingkat

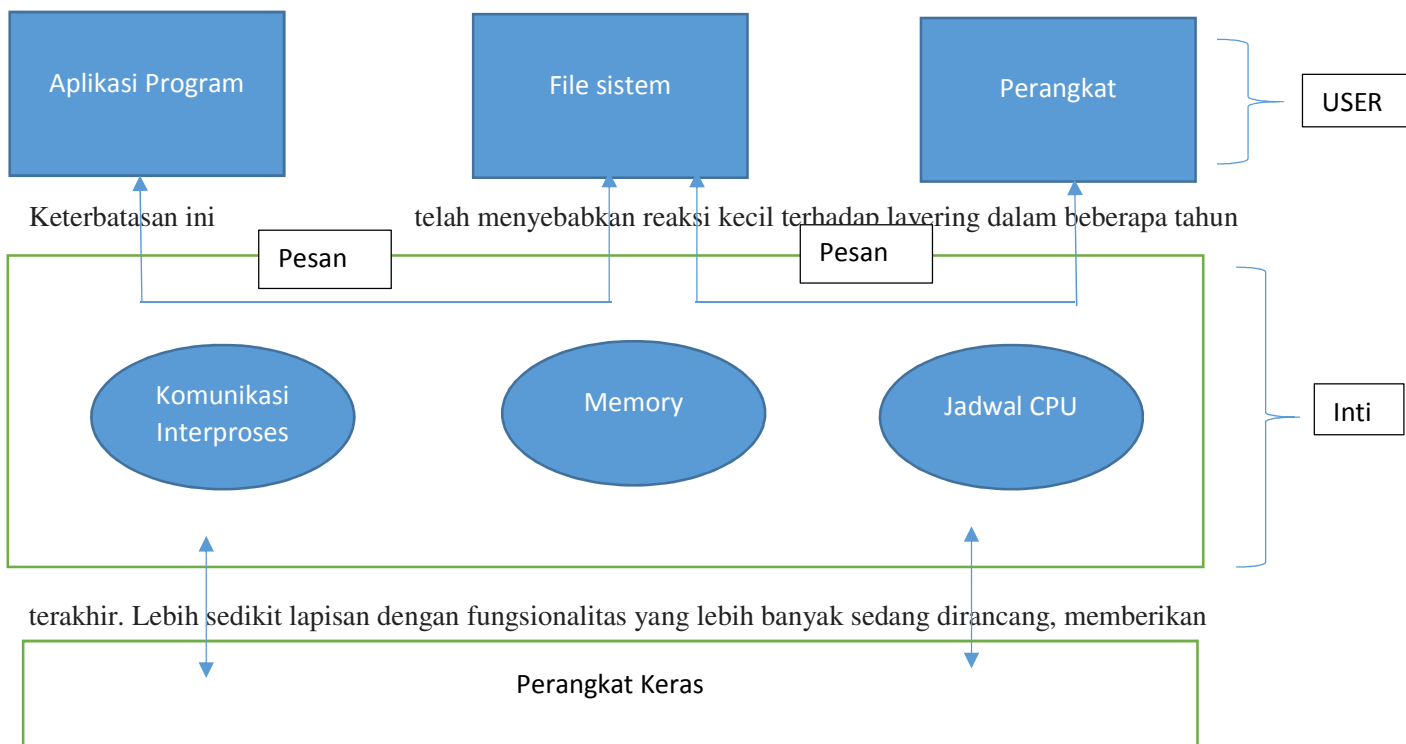
lapisan. Pendekatan ini mempermudah debug dan verifikasi sistem. Lapisan pertama bisa di-debug tanpa ada kekhawatiran untuk sisa dari sistem, karena, menurut definisi, hanya menggunakan dasar perangkat keras (yang diasumsikan benar) untuk melaksanakan fungsi-fungsinya. Setelah lapisan pertama adalah debug, fungsi yang benar dapat diasumsikan sedangkan lapisan kedua adalah debug, dan sebagainya. Jika kesalahan yang ditemukan selama debugging dari lapisan tertentu, kesalahan harus berada pada layer tersebut, karena lapisan di bawah ini yang sudah di-debug. Dengan demikian, desain dan implementasi dari sistem yang disederhanakan.

Setiap lapisan dilaksanakan hanya dengan operasi yang disediakan oleh tingkat yang lebih rendah lapisan. Lapisan tidak perlu mengetahui bagaimana operasi ini dilaksanakan; perlu tahu hanya apa operasi ini dilakukan. Oleh karena itu, setiap lapisan menyembunyikan keberadaan data tertentu struktur, operasi, dan perangkat keras dari tingkat yang lebih tinggi lapisan.

Kesulitan utama dengan pendekatan berlapis melibatkan secara tepat mendefinisikan berbagai lapisan. Karena sebuah lapisan hanya bisa menggunakan yang lebih rendah - tingkat lapisan, perencanaan yang hati-hati diperlukan. Misalnya, device driver untuk backing store (disk space yang digunakan oleh virtual-memori algoritma) harus pada tingkat yang lebih rendah dari memori-rutinitas manajemen, karena manajemen memori memerlukan kemampuan untuk menggunakan backing store. Persyaratan lain yang mungkin tidak begitu jelas. Backing-store driver biasanya akan berada di atas CPU scheduler, karena pengemudi mungkin perlu untuk menunggu I/O dan CPU dapat dijadwal ulang selama waktu ini. Namun, pada besar

sistem, penjadwal CPU mungkin memiliki lebih banyak informasi tentang semua proses aktif daripada yang dapat dimuat di memori. Oleh karena itu, informasi ini mungkin perlu ditukar masuk dan keluar dari memori, membutuhkan rutinitas driver-backing store untuk berada di bawah penjadwal CPU. Masalah terakhir dengan penerapan berlapis adalah bahwa mereka cenderung kurang efisien daripada jenis lainnya. Sebagai contoh, ketika sebuah program pengguna menjalankan operasi I / O, ia mengeksekusi sistem yang direkatkan pada lapisan I / O, yang memanggil lapisan manajemen memori, yang kemudian memanggil lapisan penjadwalan CPU, yang kemudian diteruskan ke perangkat keras. Pada masing-masing lapisan, parameter dapat dimodifikasi, datamungkin perlu disamarkan, dan begitu juga. Eachlayermenyambungkan ke pemanggilan sistem. Hasil bersihnya adalah panggilan sistem yang membutuhkan waktu lebih lama daripada yang ada pada sistem yang di-non-grup. Keterbatasan ini telah menyebabkan reaksi kecil terhadap layering dalam beberapa tahun terakhir. Lebih sedikit lapisan dengan fungsionalitas yang lebih banyak sedang dirancang, memberikan sebagian besar keuntungan dari kode termodulasi sambil menghindari masalah-masalah dari lapisan definisi dan interaksi. 2.7.3 Microkernels Kita telah melihat bahwa ketika UNIX diperluas, kernel menjadi besar dan sulit untuk dikelola. Pada pertengahan 1980-an, para peneliti di Carnegie Mellon University mengembangkan sistem operasi yang disebut Mach yang memodulasi kernel menggunakan pendekatan mikrokernel. Metode ini membentuk struktur operasi yang memunculkan komponen-komponen yang tidak penting dari sistem pelaksana negara dan sistem-tingkat program. Kernel trisisasal. Konsekuensi ada di mana yang harus disimpan di kernel dan yang harus diimplementasikan dalam ruang pengguna. Biasanya, Fungsi-fungsi utama dari sinematik antara program klien dan pelayanan yang berlalu dalam tempat berada di luar ruang. Komunikasi disediakan melalui penyampaian pesan, yang dijelaskan dalam Bagian 2.4.5. Misalnya, jika program klien ingin mengakses file, itujadwal CPU mungkin memiliki lebih banyak informasi tentang

semua proses aktif daripada yang dapat dimuat di memori. Oleh karena itu, informasi ini mungkin perlu ditukar masuk dan keluar dari memori, membutuhkan rutinitas driver-backing store untuk berada di bawah penjadwal CPU. Masalah terakhir dengan penerapan berlapis adalah bahwa mereka cenderung kurang efisien daripada jenis lainnya. Sebagai contoh, ketika sebuah program pengguna menjalankan operasi I / O, ia mengeksekusi sistem yang direkatkan pada lapisan I / O, yang memanggil lapisan manajemen memori, yang kemudian memanggil lapisan penjadwalan CPU, yang kemudian diteruskan ke perangkat keras. Pada masing-masing lapisan, parameter dapat dimodifikasi, datamungkin perlu disamarkan, dan begitu juga. Eachlayermenyambungkan ke pemanggilan sistem. Hasil bersihnya adalah panggilan sistem yang membutuhkan waktu lebih lama daripada yang ada pada sistem yang di-non-grup.



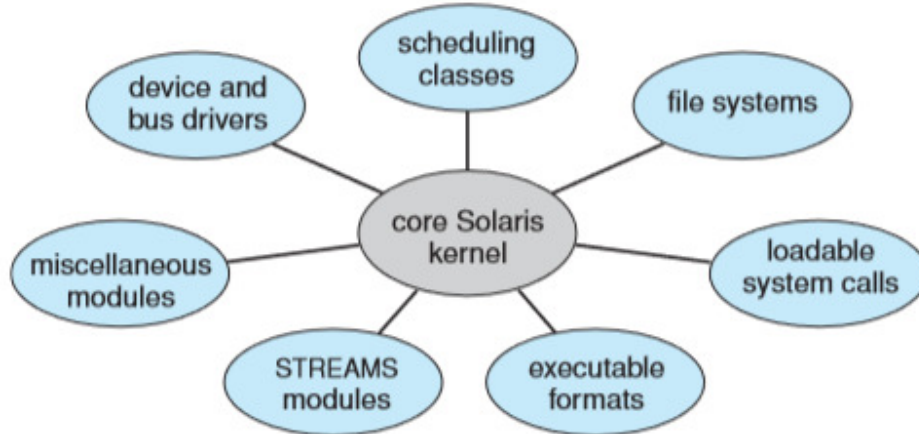
sebagian besar keuntungan dari kode termodulasi sambil menghindari masalah-masalah dari lapisan definisi dan interaksi. 2.7.3 Microkernels Kita telah melihat bahwa ketika UNIX diperluas, kernel menjadi besar dan sulit untuk dikelola. Pada pertengahan 1980-an, para peneliti di Carnegie Mellon University mengembangkan sistem operasi yang disebut Mach yang memodulasi kernel menggunakan pendekatan mikrokernel. Metode ini membentuk struktur operasi yang memunculkan komponen-komponen yang tidak penting dari sistem pelaksana negara dan sistem-tingkat program. Kernel tresiasal. Konsekuensi ada di mana yang harus disimpan di kernel dan yang harus diimplementasikan dalam ruang pengguna. Biasanya, bagaimanapun, microkernelsprovideminimalprocessandmemorymanagement, inaddition to a communication facility. Gambar 2.14 menggambarkan arsitektur mikrokernel yang khas. Fungsi-fungsi utama dari sinematikistrovoikistrointisistrikdi antaraprogramklien danpelayanan yang berlaludalam tempat berada di luar ruang. Komunikasi disediakan melalui penyampaian pesan, yang dijelaskan dalam Bagian 2.4.5. Misalnya, jika program klien ingin mengakses file,

ia harus berinteraksi dengan server file. Program klien dan kapan saja berinteraksi secara langsung. Ingat, mereka berkomunikasi secara langsung dengan mengganti pesan dengan mikrokernel. Salah satu manfaat dari pendekatan mikrokernel adalah membuatnya lebih memudahkan sistem operasi. Semua layanan baru ditambahkan ke ruang pengguna dan akibatnya tidak perlu memodifikasikan kernel. Ketika kernel harus dimodifikasi, perubahan cenderung lebih sedikit, karena mikrokernel adalah kernel yang lebih kecil. Sistem operasi yang dihasilkan lebih mudah untuk port dari satu desain perangkat keras yang lain. Mikrokernel juga menyediakan lebih banyak keamanan dan keandalan, karena sebagian besar layanan berjalan sebagai pengguna — alih-alih kernel— memproses. Jual layanan perangkat, ada yang mengoperasikan sistem yang disentuh. Beberapa sistem operasi kontemporer telah menggunakan pendekatan mikrokernel. Tru64 UNIX (sebelumnya Digital UNIX) menyediakan penggunaan antarmuka pengguna, tetapi diimplementasikan dengan kernel Mach. Peta kernel Mach Sistem UNIX memanggil pesan ke layanan tingkat pengguna yang sesuai. MacOSX kernel (juga dikenal sebagai Darwin) juga sebagian didasarkan pada mikrokernel Mach. Contoh lain adalah QNX, sistem operasi real-time untuk sistem embedded. QNX Neutrino microkernel menyediakan layanan untuk pengiriman pesan dan penjadwalan proses. Ini juga menangani komunikasi jaringan tingkat rendah dan gangguan perangkat keras. Semua layanan lain di QNX disediakan oleh proses standar yang berjalan di luar kernel in usermode. Sayangnya, kinerja pemindaian mikrokernel menderita karena peningkatan overhead fungsi

sistem. Pertimbangkan sejarah Windows NT. Rilis pertama memiliki organisasi mikrokernell berlapis. Performa versi ini rendah dibandingkan dengan Windows 95. Windows NT 4.0 memperbaiki sebagian masalah kinerja dengan memindahkan lapisan dari ruang pengguna ke ruang kernel dan mengintegrasikannya lebih dekat. Pada saat Windows XP dirancang, arsitektur Windows menjadi lebih monolitik daripada mikrokernell.

Mungkin metodologi terbaik saat ini untuk desain sistem operasi melibatkan penggunaan modul kernel yang dapat dimuat. Di sini, kernel memiliki seperangkat komponen inti dan tautan dalam layanan tambahan melalui modul, baik pada saat boot atau selama waktu berjalan. Tipe desain ini umum dalam implementasi modern sofUNIX, seperti Solaris, Linux, dan Mac OSX, serta Windows. Gagasan desain adalah untuk kernel menyediakan layanan inti sementara layanan lain diimplementasikan secara dinamis, karena kernel sedang berjalan. Layanan dinamis secara dinamis lebih baik untuk menambahkan fitur baru secara langsung ke kernel, yang akan membutuhkan kompilasi ulang kernel setiap kali ada perubahan dibuat. Jadi, misalnya, kita mungkin membangun penjadwalan CPU dan algoritma manajemen memori langsung ke kernel dan kemudian menambahkan dukungan untuk sistem file yang berbeda melalui modul yang dapat dimuat. Hasil keseluruhan menyerupai sistem berlapis dalam setiap bagian kernel yang telah diperbaiki, melindungi, tetapi memunculkan sistem berlapis, karena semua sistem mirip dengan sistem lainnya. Pendekatan ini juga mirip dengan pendekatan mikrokernell karena modul utama hanya memiliki fungsi inti dan pengetahuan tentang cara memuat dan berkomunikasi dengan modul lain; tetapi itu lebih efisien, karena modul tidak perlu memanggill pesan yang lewat di ordertocommunicate.

Tata surya adalah struktur sistem operasi, ditunjukkan, diorganisir di sekitar kernel inti dengan tujuh puluh modul beban yang dapat



dimuat:

1. Schedulingclasses
2. Filesystems
3. Loadablesystemcalls
4. Executableformats
5. STREAMS modules
6. Miscellaneous
7. Deviceand busdrivers

Linux juga menggunakan modul kernel yang dapat dimuat, terutama untuk mendukung driver perangkat dan sistem file. Kami meliputi pembuatan modul kernel yang dapat dimuat di Linux secepat programming menjalankan bagian akhir bab.

## System Hybrid

Dalam praktiknya, sangat sedikit sistem operasi yang mengadopsi struktur tunggal yang didefinisikan secara ketat. Sebagai gantinya, mereka menggabungkan struktur yang berbeda, menghasilkan sistem hibrida yang menyediakan kinerja, keamanan, dan layanan contoh, Linux dan Solaris adalah monolitik, karena memiliki sistem operasi dalam satu ruang alamat memberikan kinerja yang sangat efisien. Namun, mereka juga modular, sehingga fungsi baru dapat ditambahkan secara dinamis ke kernel. Windows juga sebagian besar monolitik (lagi-lagi terutama untuk alasan kinerja), tetapi Windows tetap memiliki perilaku khas sistem mikrokernel, termasuk menyediakan dukungan untuk subsistem terpisah (dikenal sebagai kepribadian sistem operasi) yang berjalan sebagai proses mode pengguna. Sistem Windows juga menyediakan dukungan untuk modul kernel yang dapat dimuat secara dinamis. Kami menyediakan studi kasus Linux dan Windows 7 di dalam Bab 18 dan 19, masing-masing. Di sisa bagian ini, kita mengeksplorasi struktur

Tiga sistem hibrida: sistem operasi Apple Mac OS X dan dua sistem operasi seluler paling terkemuka — iOS dan Android.

1 MacOSX Sistem operasi Apple Mac OS X menggunakan struktur hibrida., ini adalah sistem berlapis. Lapisan teratas termasuk antarmuka pengguna Aqua dan satu set lingkungan aplikasi dan layanan. Khususnya, lingkungan Cocoa menentukan API untuk bahasa pemrograman Objective-C, yang digunakan untuk menulis aplikasi Mac OS X. Di bawah lapisan ini adalah lingkungan kernel, yang terutama terdiri dari mikrokernel Mach dan kernel BSD UNIX. Mach menyediakan manajemen memori; mendukung fasilitas panggilan prosedur jarak jauh (RPC) dan interprocess communication (IPC), termasuk pengiriman pesan; dan penjadwalan benang. Komponen BSD menyediakan antarmuka baris perintah BSD, dukungan untuk jaringan dan sistem file, dan implementasi API POSIX, termasuk Pthreads. Selain Mach dan BSD, lingkungan kernel menyediakan kit I/O untuk pengembangan driver perangkat dan modul yang dapat dimuat secara dinamis (yang Mac OS X sebut sebagai ekstensi kernel)

## 2 iOS

iOS adalah sistem operasi seluler yang dirancang oleh Apple untuk menjalankan smartphone, iPhone, serta komputer tabletnya, theiPad. iOS terstruktur pada sistem operasi Mac OS X, dengan fungsionalitas tambahan yang berkaitan dengan perangkat seluler, tetapi tidak langsung menjalankan aplikasi Mac OS X. Struktur iOS muncul Cocoa Touch is an API for Objective-C that provides several frameworks untuk mengembangkan aplikasi yang berjalan di perangkat iOS. Perbedaan mendasar antara Cocoa, yang disebutkan sebelumnya, dan Cocoa Touch adalah bahwa yang terakhir memberikan dukungan untuk fitur perangkat keras yang unik untuk perangkat seluler, seperti layar sentuh. Lapisan layanan media menyediakan layanan untuk grafik, audio, dan video

### 3 Android

Sistem operasi Android dirancang oleh Open Handset Alliance (dipimpin terutama oleh Google) dan dikembangkan untuk smartphone Android dan komputer tablet. Sedangkan iOS dirancang untuk berjalan di perangkat seluler Apple dan bersumber dekat, Android berjalan di berbagai platform seluler dan bersumber terbuka, sebagian menjelaskan peningkatan popularitasnya yang cepat. Android mirip dengan iOS karena merupakan perangkat lunak berlapis yang menyediakan sekumpulan kerangka kerja yang kaya untuk mengembangkan aplikasi seluler. Di bagian bawah tumpukan perangkat lunak ini adalah kernel Linux, meskipun telah dimodifikasi oleh Google dan saat ini berada di luar distribusi normal rilis Linux

Linux digunakan terutama untuk proses, memori, dan dukungan perangkat-driver untuk perangkat keras dan telah diperluas untuk menyertakan manajemen daya. Lingkungan Android mencakup kumpulan inti perpustakaan serta mesin virtual Dalvik. Perancang perangkat lunak untuk perangkat Android mengembangkan aplikasi dalam bahasa Java. Namun, daripada menggunakan API Java standar, Google telah dirancang sebagai AndroidAPI eparate untuk Javadevelopment. File kelas Java pertama kali dikompilasi ke kode byte Java dan kemudian diterjemahkan ke dalam file yang dapat dieksekusi yang berjalan pada mesin virtual Dalvik. Mesin virtual Dalvik dirancang untuk Android dan dioptimalkan untuk perangkat seluler dengan memori terbatas dan kemampuan pemrosesan CPU. Ini untuk flibraries tersedia untuk aplikasi Android termasuk kerangka kerja untuk mengembangkan browser web (webkit), dukungan database (SQLite), dan multimedia. Liblibrary mirip dengan Clibrary standar tetapi jauh lebih kecil dan telah dirancang untuk CPU lebih lambat yang menjadi ciri perangkat mobile.

Kami telah menyebutkan sering melakukan debug dalam bab ini. Berikut, kami melihat lebih dekat. Secara global, debugging adalah aktivitas mencari dan memperbaiki kesalahan dalam sistem, baik dalam perangkat keras maupun dalam perangkat lunak. Masalah kinerja dianggap bug, sehingga debugging juga dapat menyertakan penyetulan kinerja, yang berusaha meningkatkan kinerja dengan menghapus kemacetan pemrosesan. Di bagian ini, kami menjelajahi proses debugging dan kesalahan kernel dan masalah kinerja. Debugging perangkat keras berada di luar cakupan teks ini.

Jika suatu proses gagal, sebagian besar sistem operasi menulis informasi kesalahan ke dalam log-log untuk memperingatkan para penyebar sistem operator bahwa masalah terjadi. Sistem operasi juga dapat mengambil inti dump — penangkapan memori dari proses — dan menyimpannya dalam file untuk analisis nanti. (Memori disebut sebagai "inti" pada hari-hari awal komputasi.) Menjalankan program dan dump inti dapat diperiksa oleh debugger, yang memungkinkan programmer untuk mengeksplorasi kode dan memori dari suatu proses. Debugging kode proses tingkat pengguna

Sebuah tantangan. debugging kernel sistem-operasi bahkan lebih kompleks karena ukuran dan kompleksitas kernel, kontrol atas perangkat keras, dan kurangnya alat debugging tingkat pengguna. Kegagalan di kernel disebut crash. Ketika crash terjadi, informasi kesalahan disimpan ke alog fi le, dan status memori disimpan ke crash dump. Sistem operasi debugging dan proses debugging sering menggunakan alat-alat teknik yang berbeda karena sifat yang sangat berbeda dari tugas-tugas yang ditetapkan. Pertimbangkan bahwa kegagalan kernel dalam kode sistem fi le akan membuatnya berisiko bagi kernel untuk mencoba menyimpan statusnya ke file pada sistem fi le sebelum reboot. Teknik umum adalah menyimpan status memori kernel ke bagian disk yang disisihkan untuk tujuan ini yang tidak berisi

sistem file. Jika kernel mendeteksi kesalahan yang tidak dapat diperbaiki, kernel akan menulis seluruh isi memori, atau setidaknya bagian yang dimiliki kernel dari memori sistem, ke area disk. Ketika sistem reboot, proses berjalan untuk mengumpulkan data dari itu dan menulisnya

Kami sebutkan sebelumnya bahwa penyetelan kinerja mencari untuk meningkatkan kinerja dengan menghapus kemacetan pemrosesan. Untuk mengidentifikasi bottleneck, kita harus dapat memonitor kinerja sistem. Dengan demikian, sistem operasi harus memiliki beberapa sarana komputasi dan menampilkan ukuran sehingga perilaku sistem. Dalam sejumlah sistem, sistem operasi melakukan ini dengan membuat daftar jejak sehingga perilaku sistem. Semua peristiwa menarik dicatat dengan waktu dan parameter penting mereka dan ditulis ke file. Kemudian, program analisis dapat memproses log file untuk menentukan kinerja sistem dan untuk mengidentifikasi kemacetan dan ketidakefisiensian. Jejak yang sama ini dapat dijalankan sebagai masukan untuk simulasi sistem yang disarankan yang ditingkatkan. Jejak juga dapat membantu orang untuk menemukan kesalahan dalam perilaku sistem operasi. Pendekatan lain untuk penyetelan kinerja menggunakan alat tujuan tunggal dan interaktif yang memungkinkan pengguna dan administrator untuk mempertanyakan keadaan berbagai komponen sistem untuk mencari leher botol. Satu hal seperti itu juga menyebabkan perintah UNIX untuk menampilkan sumber daya yang digunakan pada sistem, juga sebagai daftar yang diurutkan dari proses penggunaan sumber daya "teratas". Alat-alat lain menampilkan keadaan disk I/O, lokasi memori, dan jaringan trafik. Windows Task Manager adalah alat yang serupa untuk sistem Windows. Manajer tugas mencakup informasi untuk aplikasi saat ini serta proses, penggunaan CPU dan memori, dan statistik jaringan. Membuat sistem operasi lebih mudah untuk dipahami, dideteksi, dan diperbaiki saat mereka menjalankan adalah area aktif penelitian dan implementasi. Generasi baru alat analisis kinerja yang didukung oleh kernel telah membuat perbaikan yang signifikan dalam bagaimana tujuan ini dapat dicapai

### Tuning kinerja

Kami sebutkan earlierthat kinerja tuning berusaha untuk meningkatkan kinerja dengan menghapus pengolahan kemacetan. Untuk mengidentifikasi kemacetan, kita harus mampu untuk memantau kinerja sistem. Dengan demikian, sistem operasi harus memiliki beberapa sarana komputasi dan menampilkan mengukur sehingga perilaku sistem. Dalam sejumlah sistem, sistem operasi melakukan hal ini dengan memproduksi melacak daftar perilaku sistem. Semua peristiwa-peristiwa yang menarik dicatat dengan waktu mereka dan parameter-parameter yang penting dan ditulis ke file. Kemudian, sebuah program analisis proses file log untuk menentukan kinerja sistem dan untuk mengidentifikasi hambatan dan efisiensi. Ini jejak yang sama dapat dijalankan sebagai masukan untuk simulasi menyarankan perbaikan sistem. Jejak juga dapat membantu orang untuk menemukan kesalahan dalam operasi-sistem perilaku. Pendekatan lain untuk tuning kinerja menggunakan satu tujuan, alat-alat interaktif yang memungkinkan pengguna dan administrator untuk mempertanyakan keadaan berbagai komponen sistem untuk mencari kemacetan. Salah satu alat tersebut mempekerjakan theUNIX komando atas untuk menampilkan sumber daya yang digunakan pada sistem, serta daftar diurutkan dari "atas" sumber daya menggunakan proses. Alat-alat lain yang menampilkan keadaan disk I/O, alokasi memori, dan jaringan lalu lintas. Windows Task Manager adalah sebuah alat serupa untuk sistem Windows. Tugas manajer meliputi informasi untuk saat ini aplikasi serta proses-proses, penggunaan CPU dan memori, dan jaringan statistik. Membuat sistem operasi lebih mudah untuk memahami, debug, dan tune seperti yang mereka jalankan adalah aplikasi yang aktif di bidang penelitian dan implementasi. Generasi baru dari kernel-enabled kinerja alat



analisis telah membuat perbaikan yang signifikan dalam bagaimana tujuan ini dapat dicapai. Berikutnya, kita akan membahas sebuah contoh utama dari alat tersebut: Solaris 10 Dynamic Tracing Fasilitas.

## DTrace

DTrace merupakan fasilitas yang secara dinamis menambahkan probe untuk sistem yang berjalan, baik dalam proses pengguna dan kernel. Probe ini dapat dilihat melalui D bahasa pemrograman untuk menentukan jumlah yang menakutkan tentang kernel, sistem negara, dan proses kegiatan. Debugging interaksi antara user-level dan kernel code adalah hampir mustahil tanpa toolset yang memahami kedua set kode dan alat interaksi. Untuk itu toolset untuk menjadi benar-benar berguna, itu harus mampu untuk men-debug daerah manapun dari sistem, termasuk daerah yang tidak ditulis dengan debugging dalam pikiran, dan melakukannya tanpa mempengaruhi keandalan sistem. Alat ini juga harus memiliki minimal kinerja dampak—pilihan itu seharusnya tidak memiliki dampak selama penggunaan. The DTrace tool memenuhi persyaratan ini dan memberikan yang dinamis, aman, rendah dampak debugging lingkungan. Sampai DTrace kerangka kerja dan alat-alat menjadi tersedia dengan Solaris 10, kernel debugging biasanya diselubungi misteri dan dicapai melalui terdapat sikap dan kuno kode dan alat-alat. Misalnya, Cpu mempunyai breakpoint fitur yang akan menghentikan eksekusi dan memungkinkan debugger untuk memeriksa keadaan dari sistem. Kemudian eksekusi dapat berlanjut sampai keesokan breakpoint or penghentian. Metode ini tidak dapat digunakan secara multiuser sistem operasi kernel tanpa negatif affectin empedu pengguna pada sistem. Profil, yang secara berkala sampel instruksi pointer untuk menentukan kode yang sedang dieksekusi, dapat menunjukkan statistik tren tapi tidak setiap kegiatan. Kode dapat dimasukkan dalam kernel untuk memancarkan data tertentu di bawah kondisi tertentu, tetapi kode yang memperlambat kernel dan cenderung untuk tidak dimasukkan dalam bagian dari kernel mana masalah tertentu debugged terjadi.

Bertentangan, DTrace berjalan pada sistem produksi yang berjalan penting atau aplikasi penting—dan tidak menyebabkan kerusakan pada sistem. Memperlambat kegiatan sementara diaktifkan, tapi setelah eksekusi ini me-reset sistem pra-debugging negara. Hal ini juga yang luas dan mendalam alat. Hal ini dapat secara luas debug segala sesuatu yang terjadi dalam sistem (baik pada pengguna dan kernel level dan antara pengguna dan kernel lapisan). Hal ini juga dapat menggali jauh ke dalam kode, yang menunjukkan individu CPU instruksi atau kernel subrutin kegiatan. DTrace terdiri dari compiler, kerangka kerja, penyedia probe tertulis dalam kerangka itu, dan konsumen mereka probe. DTrace penyedia membuat probe. Struktur Kernel yang ada untuk melacak semua probe bahwa penyedia mencukur dibuat. Probe disimpan dalam hash table struktur data yang dirinci berdasarkan nama dan diindeks menurut unik probe pengenalan. Ketika probe diaktifkan, sedikit kode di area yang akan diperiksa ulang untuk memanggil dtrace probe (probe identifier) dan kemudian melanjutkan dengan kode awal operasi. Penyedia yang berbeda membuat berbagai jenis probe. Sebagai contoh, sebuah kernel sistem-panggilan probe bekerja dengan cara yang berbeda dari user-proses penyelidikan, dan yang berbeda dari I/O probe. DTrace memiliki compiler yang menghasilkan kode byte yang berjalan di kernel. Kode ini dipastikan akan "aman" oleh compiler. Misalnya, tidak ada loop yang diizinkan, dan hanya spesifik kernel negara modifikasi diijinkan bila secara khusus meminta. Hanya pengguna dengan DTrace "hak istimewa" (atau "root" user) diperbolehkan untuk menggunakan DTrace, karena dapat mengambil pribadi kernel data (dan memodifikasi data jika diminta). Kode yang dihasilkan berjalan di kernel dan memungkinkan probe. Hal ini juga memungkinkan konsumen dalam mode pengguna dan memungkinkan komunikasi antara dua. Sebuah DTrace konsumen adalah kode yang tertarik dalam penyelidikan dan hasil-hasilnya. Konsumen meminta agar penyedia membuat satu atau lebih probe. Ketika probe kebakaran,

memancarkan data yang dikelola oleh kernel. Di dalam kernel, tindakan yang disebut memungkinkan kontrol blok, atau ECBs, dilakukan ketika probe api. Satu probe dapat menyebabkan beberapa ECBs untuk mengeksekusi jika lebih dari satu konsumen lebih tertarik dalam penyelidikan. Masing-masing ECB berisi predikat ("jika pernyataan") yang dapat menyaring bahwa ECB. OOtherwisethe daftar tindakan ECB dijalankan. Yang paling umum adalah tindakan untuk menangkap beberapa bit data, seperti variabel nilai pada titik probe execution. By pengumpulan data tersebut.

sistem tidak memiliki perjanjian lisensi yang saling bertentangan. Misalnya, Dtrace ditambahkan ke Mac OS X dan FreeBSD dan kemungkinan akan menyebar lebih jauh karena itu kemampuan unik. Sistem operasi lain, khususnya turunan Linux, menambahkan fungsionalitas pelacakan-kernel juga. sistem operasi lain mulai memasukkan alat kinerja dan penelusuran yang dipupuk oleh penelitian di berbagai institusi, termasuk proyek Paradyn.

## 2.9 GENERASI SISTEM OPERASI

Ada kemungkinan untuk merancang, mengkode, dan menerapkan sistem operasi secara khusus untuk satu mesin di satu situs. Lebih umum lagi, bagaimanapun, sistem operasi dirancang untuk berjalan di salah satu kelas mesin di berbagai situs dengan berbagai konfigurasi perifer. Sistem kemudian harus dikonfigurasi atau dihasilkan untuk setiap situs komputer tertentu, suatu proses yang kadang-kadang dikenal sebagai system generation SYSGEN.

Sistem operasi terdistribusi secara normal pada disk, CD-ROM atau DVD-ROM, atau sebagai "ISO" gambar, yang merupakan file dalam format CD-ROM atau DVD-ROM. Untuk menghasilkan suatu sistem, kita menggunakan program khusus. Ini SYSGEN program membaca dari file yang diberikan, atau meminta operator sistem untuk informasi mengenai konfigurasi khusus dari sistem perangkat keras, atau probe hardware secara langsung untuk menentukan komponen apa yang ada. Yang berikut ini jenis informasi yang harus ditentukan.

- Apa fungsi CPU? Apa pilihan (extended instruction set, floating point aritmatika, dan sebagainya) yang dipasang? Untuk beberapa CPU sistem, masing-masing CPU dapat digambarkan
- Bagaimana akan boot disk yang akan diformat? Berapa banyak bagian, atau "partisi" ia akan dipisahkan ke dalam, dan apa yang akan masuk ke masing-masing partisi?
- Berapa banyak memori yang tersedia? Beberapa sistem akan menentukan nilai ini sendiri dengan referensi lokasi memori setelah lokasi memori sampai "illegal alamat" kesalahan yang dihasilkan. Prosedur ini mendefinisikan hukum akhir alamat dan karenanya jumlah memori yang tersedia.
- Perangkat apa yang tersedia? Sistem akan perlu untuk mengetahui bagaimana untuk mengatasi setiap perangkat (device), perangkat mengganggu jumlah, perangkat jenis dan model, dan setiap perangkat khusus karakteristik.
- Apa sistem operasi pilihan yang diinginkan, atau apa nilai parameter untuk digunakan? Ini pilihan atau nilai-nilai yang mungkin termasuk berapa banyak buffer yang ukuran yang harus digunakan, apa jenis CPU-algoritma penjadwalan adalah yang diinginkan, apa yang maksimum jumlah dari proses yang akan didukung, dan sebagainya.

Setelah informasi ini ditentukan, hal ini dapat digunakan dalam beberapa cara. Di salah satu ekstrim, administrator sistem dapat menggunakannya untuk memodifikasi salinan dari kode sumber sistem operasi. Sistem operasi kemudian adalah benar-benar disusun. Data deklarasi, inisialisasi, dan konstanta, bersama dengan kompilasi bersyarat, menghasilkan output-objek

versi sistem operasi yang disesuaikan dengan sistem yang dijelaskan. Pada sedikit kurang disesuaikan tingkat, deskripsi sistem dapat menyebabkan pembuatan tabel dan pemilihan modul dari precompiled perpustakaan. Modul ini dihubungkan bersama untuk membentuk dihasilkan sistem operasi. Pilihan ini memungkinkan perpustakaan untuk memuat driver perangkat yang didukung untuk semua I/O perangkat, tetapi hanya orang-orang yang diperlukan dihubungkan ke sistem operasi. Karena sistem ini tidak dikompilasi ulang, sistem generasi yang lebih cepat, tetapi yang dihasilkan sistem mungkin terlalu umum. Pada ekstrem yang lain, adalah mungkin untuk membangun sebuah sistem yang benar-benar meja didorong. Semua kode ini selalu menjadi bagian dari sistem, dan pilihan yang terjadi di waktu pelaksanaan, bukan pada saat kompilasi atau link. Sistem generasi melibatkan hanya menciptakan tabel yang sesuai untuk menggambarkan sistem. Perbedaan utama antara pendekatan ini adalah ukuran dan umum yang dihasilkan sistem dan kemudahan memodifikasi itu sebagai hardware perubahan konfigurasi. Mempertimbangkan biaya memodifikasi sistem untuk mendukung baru diperoleh grafis terminal atau disk drive. Seimbang terhadap yang biaya, tentu saja, adalah frekuensi (atau infrequency) dari perubahan tersebut.

Setelah sistem operasi yang dihasilkan, harus dibuat tersedia untuk digunakan oleh hardware. Tapi bagaimana hardware tahu di mana kernel atau cara beban itu kernel? Prosedur dari mulai komputer dengan memuat kernel dikenal sebagai booting sistem. Pada sebagian besar sistem komputer, sepotong kecil kode dikenal sebagai bootstrap program bootstrap loader menempatkan kernel, beban itu ke memori utama, dan dimulai pelaksanaannya. Beberapa sistem komputer, seperti Pc, menggunakan proses dua langkah yang sederhana bootstrap loader menjemput yang lebih kompleks program boot dari disk, yang pada gilirannya beban kernel. Ketika CPU menerima reset event—misalnya, ketika dinyalakan atau reboot—instruksi register yang penuh dengan memori yang telah ditetapkan lokasi, dan eksekusi dimulai di sana. Di lokasi itu adalah awal bootstrap program. Program ini adalah dalam bentuk read-only memory (ROM), karena RAM adalah dalam sebuah negara tidak diketahui pada sistem startup. ROM nyaman karena hal itu tidak membutuhkan inisialisasi dan tidak mudah terinfeksi oleh virus komputer. Bootstrap program yang dapat melakukan berbagai tugas. Biasanya, salah satu tugas untuk menjalankan diagnostik untuk menentukan keadaan mesin. Jika diagnose lulus, program ini dapat terus berlanjut dengan booting langkah-langkah. Hal ini juga dapat menginisialisasi semua aspek-aspek dari sistem, dari CPU register untuk perangkat pengendali dan isi dari memori utama. Cepat atau lambat, mulai dari sistem operasi. Beberapa sistem—seperti telepon seluler, tablet, dan konsol game—took seluruh sistem operasi di ROM. Menyimpan sistem operasi di ROM cocok untuk usaha kecil sistem operasi, sederhana mendukung perangkat keras, dan kasar operasi. Masalah dengan pendekatan ini adalah bahwa mengubah kode bootstrap membutuhkan mengubah ROM chip hardware. Beberapa sistem mengatasi masalah ini dengan menggunakan erasable programmable read-only memory (EPROM), yang readonly kecuali bila secara eksplisit diberikan perintah untuk menjadi writable. Semua bentuk ROM juga dikenal sebagai firmware, karena karakteristik mereka jatuh di suatu tempat antara orang-orang dari hardware dan software. Masalah dengan firmware secara umum adalah bahwa mengeksekusi kode tidak lebih lambat dari yang mengeksekusi kode di RAM. Beberapa sistem menyimpan sistem operasi di firmware dan copy ke RAM untuk eksekusi cepat. Akhir masalah dengan firmware adalah bahwa hal itu relatif mahal, sehingga biasanya hanya dalam jumlah kecil yang tersedia. Untuk sistem operasi (termasuk operasi tujuan umum sistem seperti Windows, Mac OS X, dan UNIX) atau untuk sistem yang berubah sering, bootstrap loader disimpan dalam firmware, dan sistem operasi pada disk. Dalam hal ini, bootstrap menjalankan diagnostik dan memiliki sedikit kode yang dapat membaca satu blok di lokasi yang tetap (misalnya blok nol) dari disk ke memori dan mengeksekusi kode dari yang boot block. Program yang tersimpan di blok boot dapat menjadi cukup canggih untuk memuat

seluruh sistem operasi ke memori dan memulai pelaksanaannya. Lebih biasanya, ini adalah kode sederhana (karena cocok dalam satu blok disk) dan yang tahu hanya alamat pada disk dan panjang sisa bootstrap program. GRUB adalah contoh dari sebuah open-source bootstrap program untuk sistem Linux. Semua disk yang terikat bootstrap, dan sistem operasi itu sendiri, dapat dengan mudah diubah dengan menulis versi baru ke disk. Disk yang memiliki partisi boot (lebih pada bahwa dalam Pasal 10.5.1) disebut boot disk atau system disk. Sekarang yang penuh bootstrap program yang telah dimuat, dapat melintasi file sistem untuk menemukan kernel sistem operasi, memuatnya ke dalam memori, dan memulai pelaksanaannya. Hanya pada titik ini bahwa sistem ini dikatakan berjalan

Sistem operasi menyediakan sejumlah layanan. Pada tingkat terendah, sistem panggilan memungkinkan suatu program yang sedang berjalan untuk membuat permintaan dari sistem operasi ini diatas. Pada tingkat yang lebih tinggi, command interpreter atau shell menyediakan mekanisme bagi pengguna untuk mengeluarkan permintaan tanpa menulis program. Perintah dapat berasal dari file selama batch-mode eksekusi atau secara langsung dari terminal atau desktop GUI ketika interaktif atau waktu-mode bersama. Program sistem disediakan untuk memenuhi banyak permintaan pengguna. Jenis permintaan bervariasi menurut tingkat. Sistem-panggilan level harus menyediakan fungsi-fungsi dasar, seperti kontrol proses dan file dan perangkat manipulasi. Tingkat yang lebih tinggi permintaan, puas oleh command interpreter atau sistem program, yang diterjemahkan ke dalam urutan sistem panggilan. Sistem pelayanan dapat diklasifikasikan menjadi beberapa kategori: kontrol program, status permintaan, dan I/O permintaan. Kesalahan Program dapat dianggap implisit permintaan untuk layanan. Desain baru sistem operasi adalah tugas utama. Adalah penting bahwa dari sistem dapat didefinisikan dengan baik sebelum desain dimulai. Jenis sistem yang diinginkan adalah dasar untuk pilihan-pilihan di antara berbagai algoritma dan strategi yang akan dibutuhkan. Sepanjang seluruh siklus desain, kita harus berhati-hati untuk memisahkan kebijakan keputusan dari rincian pelaksanaan (mekanisme). Pemisahan ini memungkinkan fleksibilitas maksimum jika keputusan-keputusan kebijakan yang akan diubah nanti. Setelah sistem operasi ini dirancang, itu harus dilaksanakan. Operasi sistem ini hampir selalu ditulis dalam sistem-implementasi bahasa atau di tingkat yang lebih tinggi itu. Fitur ini meningkatkan pelaksanaannya, pemeliharaan, dan portabilitas. Sebuah sistem yang besar dan kompleks seperti sistem operasi modern harus dapat direkayasa dengan hati-hati. Modularitas adalah penting. Merancang sistem sebagai urutan lapisan atau menggunakan microkernel dianggap sebagai teknik yang baik. Banyak sistem operasi yang sekarang mendukung dimuat secara dinamis modul, yang memungkinkan menambahkan fungsi untuk sistem operasi saat ini mengeksekusi. Umumnya, sistem operasi yang mengadopsi pendekatan hybrid yang menggabungkan beberapa berbeda jenis struktur. Proses Debugging dan kernel kegagalan dapat dicapai melalui menggunakan debugger dan alat lain yang menganalisis core dump. Alat-alat seperti DTrace menganalisis sistem produksi untuk menemukan kemacetan dan memahami sistem lainnya perilaku. Untuk membuat sistem operasi untuk suatu konfigurasi mesin tertentu, kita harus melakukan system generation. Untuk sistem komputer untuk mulai berjalan, CPU harus menginisialisasi dan mulai menjalankan bootstrap program dalam firmware. Bootstrap dapat menjalankan sistem operasi secara langsung jika sistem operasi juga di dalam firmware, atau dapat menyelesaikan urutan di mana beban semakin pintar program dari firmware dan disk sampai operasi sistem itu sendiri dimuat ke memori dan dieksekusi.

## Latihan Praktek

2.1 Apa tujuan dari system calls?

2.2 Apa saja lima kegiatan utama dari sistem operasi yang berkaitan dengan

proses manajemen?

2.3 apakah tiga kegiatan utama dari sistem operasi yang berkaitan dengan untuk manajemen memori?

2.4 apakah tiga kegiatan utama dari sistem operasi yang berkaitan dengan ke secondary-storage management?

2.5 Apa tujuan dari command interpreter? Mengapa biasanya terpisah dari kernel?

Latihan 95

2.6 sistem Apa yang memiliki panggilan untuk dieksekusi oleh command interpreter atau shell

dalam rangka untuk memulai proses baru?

2.7 Apa tujuan dari sistem program?

2.8 Apa keuntungan utama dari pendekatan berlapis untuk desain sistem?

Apa kelemahan dari pendekatan berlapis?

2.9 Daftar lima layanan yang disediakan oleh sistem operasi, dan menjelaskan bagaimana masing-masing

menciptakan kenyamanan bagi pengguna. Di mana kasus-kasus itu akan menjadi mustahil untuk

user-level program-program untuk menyediakan layanan ini? Jelaskan jawaban anda.

2.10 Mengapa beberapa sistem menyimpan sistem operasi di firmware, sementara lain-lain menyimpannya pada disk?

2.11 Bagaimana bisa sebuah sistem yang akan dirancang untuk memungkinkan pilihan sistem operasi

yang boot? Apa yang akan bootstrap program yang perlu dilakukan?

Latihan

2.12 layanan dan fungsi yang disediakan oleh sistem operasi dapat dibagi menjadi dua kategori utama. Jelaskan secara singkat dua kategori, dan mendiskusikan bagaimana mereka berbeda.

2.13 Menggambarkan tiga metode umum untuk melewati parameter untuk operasi sistem.

2.14 Menjelaskan bagaimana anda bisa mendapatkan profil statistik jumlah waktu dikeluarkan oleh sebuah program yang mengeksekusi berbagai bagian dari kode.

Membahas

pentingnya seperti mendapatkan profil statistik.

2.15 Apa lima kegiatan utama dari sistem operasi yang berkaitan dengan manajemen file?

2.16 Apa keuntungan dan kerugian menggunakan yang sama systemcall antarmuka untuk memanipulasi file dan perangkat?

2.17 itu Akan mungkin bagi pengguna untuk mengembangkan sebuah command interpreter

menggunakan system-call interface yang disediakan oleh sistem operasi?

2.18 Apa dua model komunikasi interprocess? Apa kekuatan dan kelemahan dari dua pendekatan?

2.19 Mengapa pemisahan antara mekanisme dan kebijakan yang diinginkan?

2.20 Hal ini kadang-kadang sulit untuk mencapai pendekatan berlapis jika dua komponen

sistem operasi bergantung pada satu sama lain. Mengidentifikasi skenario di mana hal ini tidak jelas bagaimana untuk lapisan kedua komponen sistem yang memerlukan

kopling ketat dari fungsi mereka.

2.21 Apa keuntungan utama dari pendekatan mikrokernel untuk sistem desain? Bagaimana program-program pengguna dan sistem pelayanan yang berinteraksi dalam

mikrokernel arsitektur? Apa kerugian menggunakan pendekatan mikrokernel?

2.22 Apa keuntungan dari menggunakan kernel loadable modules?

96 Bab 2 Operasi-Sistem Struktur

2.23 Bagaimana iOS dan Android mirip? Bagaimana mereka berbeda?

2.24 Menjelaskan mengapa program Java yang berjalan pada sistem Android tidak menggunakan

standar Java API dan mesin virtual.

2.25 percobaan Sintesis sistem operasi memiliki sebuah assembler dimasukkan dalam kernel. Untuk mengoptimalkan sistem kinerja panggilan, kernel merakit rutinitas dalam ruang kernel untuk meminimalkan jalan yang system call harus mengambil melalui kernel. Pendekatan ini adalah antitesis dari pendekatan berlapis, di mana jalan melalui kernel diperpanjang untuk membuat bangunan sistem operasi lebih mudah. Membahas pro dan kontra Sintesis pendekatan untuk desain kernel dan sistem kinerja optimasi.

2.26 Dalam Bagian 2.3, kita dijelaskan sebuah program yang menyalin isi dari satu file untuk file tujuan. Program ini bekerja dengan terlebih dahulu mendorong pengguna untuk nama file sumber dan tujuan. Menulis program ini menggunakan baik Windows atau POSIX API. Pastikan untuk mencakup semua yang diperlukan kesalahan memeriksa, termasuk memastikan bahwa sumber file yang ada. Setelah anda telah benar-benar dirancang dan diuji program ini, jika anda digunakan sistem yang mendukung hal ini, menjalankan program dengan menggunakan utilitas yang jejak sistem panggilan. Sistem Linux menyediakan strace utilitas, dan Solaris dan Mac OS X sistem menggunakan dtrace perintah. Sebagai sistem Windows melakukan tidak menyediakan fitur tersebut, anda akan memiliki untuk melacak melalui Windows versi program ini menggunakan debugger Linux Kernel Modules

Dalam proyek ini, anda akan belajar bagaimana untuk membuat modul kernel dan beban itu ke Kernel Linux. Proyek dapat diselesaikan dengan menggunakan mesin virtual Linux yang tersedia dengan teks ini. Meskipun anda dapat menggunakan editor untuk menulis ini C program-program, anda akan memiliki untuk menggunakan aplikasi terminal untuk mengkompilasi program, dan anda akan memiliki untuk memasukkan perintah pada command line untuk mengelola modul dalam kernel. Seperti yang akan anda temukan, keuntungan dari mengembangkan modul kernel adalah bahwa hal itu adalah relatif mudah metode berinteraksi dengan kernel, sehingga memungkinkan anda untuk menulis program-program yang langsung memanggil fungsi kernel. Hal ini penting untuk anda untuk menjaga dalam pikiran bahwa anda memang menulis kernel code yang secara langsung berinteraksi dengan kernel. Yang biasanya berarti bahwa setiap kesalahan dalam kode bisa crash sistem! Namun, karena anda akan menggunakan mesin virtual, setiap kegagalan yang memburuk hanya memerlukan reboot sistem.

Bagian I—Membuat Modul Kernel

Bagian pertama dari proyek ini melibatkan serangkaian langkah-langkah untuk menciptakan dan memasukkan modul ke dalam kernel Linux. Anda bisa daftar semua modul kernel yang saat ini sarat dengan memasukkan perintah

```
lsmod
```

Perintah ini akan menampilkan daftar saat ini modul kernel dalam tiga kolom: nama, size, dan mana modul yang digunakan. Program berikut (bernama sederhana.c dan tersedia dengan sumber kode untuk teks ini) menggambarkan yang sangat dasar modul kernel yang sesuai cetakan pesan ketika modul kernel yang dimuat dan dibongkar.

```
#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/module.h>

/* This function is called when the module is loaded. */
int simple_init(void)
{
    printk(KERN_INFO "Loading Module\n");
    return 0;
}

/* This function is called when the module is removed. */
void simple_exit(void)
{
    printk(KERN_INFO "Removing Module\n");
}

/* Macros for registering module entry and exit points. */
module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("SGG");
```

Fungsi sederhana init() adalah modul entry point, yang merupakan fungsi yang dipanggil saat modul dimuat ke dalam kernel. Demikian pula, sederhana exit() fungsi adalah modul titik keluar— fungsi yang dipanggil saat modul dihapus dari kernel.

Modul entry point fungsi harus mengembalikan nilai integer, dengan 0 mewakili keberhasilan dan nilai lain yang mewakili kegagalan. Modul exit point fungsi kembali batal. Baik modul entry point atau modul titik keluar dilewatkan parameter apapun. Berikut dua macro yang digunakan untuk mendaftarkan modul entry dan exit point dengan kernel:

```
Module_init()
```

```
Module_exit()
```

Perhatikan bagaimana kedua modul entry dan exit point fungsi melakukan panggilan untuk printk() fungsi. printk() adalah kernel setara dengan printf(), namun outputnya dikirim ke kernel

log buffer yang isinya dapat dibaca oleh perintah `dmesg`. Salah satu perbedaan antara `printf()` dan `printk()` adalah bahwa `printk()` memungkinkan kita untuk menentukan prioritas bendera dan nilai-nilai yang diberikan dalam `<linux/printk.h>` sertakan file. Dalam hal ini, prioritas adalah `KERN INFO`, yang didefinisikan sebagai pesan informasi. Akhir garis—`MODUL LISENSI()`, `DESKRIPSI MODUL()`, dan `MODUL PENULIS()`—mewakili rincian mengenai lisensi perangkat lunak, deskripsi modul, dan penulis. Untuk tujuan kita, kita tidak bergantung pada informasi ini, tapi kita seperti itu karena itu adalah praktek standar dalam mengembangkan modul kernel. Modul kernel ini `sedherhana.c` disusun dengan menggunakan `Makefile` yang menyertainya kode sumber dengan proyek ini. Untuk mengkompilasi modul, masukkan berikut pada command line:

```
make
```

Kompilasi menghasilkan beberapa file. File `sedherhana.ko` mewakili disusun modul kernel. Berikut langkah menggambarkan memasukkan modul ini ke dalam kernel Linux Modul Kernel yang dimuat menggunakan `insmod` perintah, yang dijalankan sebagai berikut: `sudo insmod simple.ko` Untuk memeriksa apakah modul telah dimuat, masukkan perintah `lsmod` dan pencarian untuk modul `sedherhana`. Ingat bahwa modul entry point dipanggil ketika modul dimasukkan ke dalam kernel. Untuk memeriksa isi dari pesan ini di kernel log buffer, masukkan perintah

```
dmesg
```

Anda akan melihat pesan "Loading Module." Menghapus modul kernel yang melibatkan menyerukan `rmmod` perintah (perhatikan bahwa `.ko` akhiran yang tidak perlu):

```
sudo rmmod simple
```

Pastikan untuk memeriksa dengan perintah `dmesg` untuk memastikan modul telah dihapus. Karena kernel log buffer dapat mengisi dengan cepat, sering masuk akal untuk jelas buffer secara berkala. Hal ini dapat dicapai sebagai berikut:

```
sudo dmesg -c
```

## Bagian I Tugas

Lanjutkan melalui langkah-langkah yang dijelaskan di atas untuk membuat modul kernel dan untuk memuat dan membongkar modul. Pastikan untuk memeriksa isi dari log kernel buffer menggunakan `dmesg` untuk memastikan anda telah benar mengikuti langkah-langkah.

## Bagian II—Data Kernel Struktur

Bagian kedua dari proyek ini melibatkan memodifikasi modul kernel sehingga menggunakan kernel linked-list struktur data. Dalam Bagian 1.10, kita membahas berbagai struktur data yang umum di sistem operasi. Kernel Linux menyediakan beberapa struktur ini. Di sini, kami mengeksplorasi menggunakan melingkar, doubly linked list yang tersedia untuk kernel pengembang. Banyak dari apa yang kita bahas adalah yang tersedia di Linux source code—dalam contoh ini, file include `<linux/daftar.h>`—dan kami menyarankan anda memeriksa file ini sebagai anda melanjutkan melalui langkah-langkah berikut. Awalnya, anda harus menentukan sebuah struct yang mengandung unsur-unsur yang harus dimasukkan ke dalam linked list. Berikut C struct mendefinisikan tanggal lahir:

```
struct birthday {  
    int day;  
    int month;
```



```
int year;
struct list_head list;
}
```

Perilaku anggota struct daftar kepala daftar. Daftar kepala struktur didefinisikan dalam file include <linux/jenis.h>. Niatnya adalah untuk menanamkan linked list di dalam node yang terdiri dari daftar. Ini daftar kepala struktur cukup sederhana—itu hanya memiliki dua anggota, next dan prev, yang mengarah ke berikutnya dan sebelumnya entri dalam daftar. Dengan menanamkan linked list dalam struktur, Linux memungkinkan untuk mengelola data-data struktur dengan serangkaian fungsi makro Memasukkan Elemen ke dalam Linked List Kita dapat mendeklarasikan sebuah daftar kepala objek, yang kita gunakan sebagai referensi untuk kepala daftar dengan menggunakan LIST\_HEAD() makro static LIST\_HEAD(birthday\_list); Makro ini mendefinisikan dan menginisialisasi variabel daftar ulang, yang merupakan jenis struct list\_head Kami membuat dan menginisialisasi contoh struct ulang tahun sebagai berikut:

```
struct birthday *person;

person = kmalloc(sizeof(*person), GFP_KERNEL);

person->day = 2;

person->month = 8;

person->year = 1995;

INIT_LIST_HEAD(&person->list);
```

Yang kmalloc() fungsi kernel setara tingkat pengguna malloc()

fungsi untuk mengalokasikan memori, kecuali bahwa kernel memori yang dialokasikan. (GFP KERNEL bendera menunjukkan rutin kernel alokasi memori.) Makro INIT DAFTAR KEPALA() akan menginisialisasi daftar anggota dalam struct ulang tahun. Kita dapat kemudian tambahkan ini misalnya untuk akhir linked list menggunakan daftar add tail() macro:

```
list_add_tail(&person->list, &birthday list);
```

mentransfer linked list

Melintasi daftar melibatkan menggunakan list\_for\_each\_entry() Makro, yang menerima tiga parameter:

- Pointer ke struktur yang sedang melakukan iterasi atas
- Pointer ke kepala daftar menjadi iterasi lebih dari
- Nama variabel yang berisi daftar struktur kepala\

Kode berikut menggambarkan makro ini:

```
struct birthday *ptr;

list_for_each_entry(ptr, &birthday list, list) {
```

```
/* on each iteration ptr points */  
/* to the next birthday struct */  
}
```

## Menghapus Elemen dari Linked List

Menghapus elemen dari daftar yang melibatkan menggunakan daftar `del()` makro, yang melewati sebuah pointer ke struct `list_head`

```
List_del(struct list_head *element)
```

ini menghapus elemen dari daftar sementara mempertahankan struktur sisa dari daftar. Mungkin pendekatan yang paling sederhana untuk menghapus semua elemen dari linked list adalah untuk menghapus setiap elemen seperti yang anda traverse daftar. Makro `list_for_each_entry_safe()` bersifat sama seperti `list_for_each_entry()` kecuali bahwa itu berlalu argumen tambahan yang mempertahankan nilai pointer berikutnya item yang dihapus. (Hal ini diperlukan untuk menjaga struktur dari daftar.) Contoh kode berikut menggambarkan makro ini:

```
struct birthday *ptr, *next  
  
list_for_each_entry_safe(ptr,next,&birthday list,list) {  
/* on each iteration ptr points */  
/* to the next birthday struct */  
  
list_del(&ptr->list);  
kfree(ptr);  
}
```

Perhatikan bahwa setelah menghapus setiap elemen, kita kembali memori yang sebelumnya dialokasikan dengan `kmalloc()` kembali ke kernel dengan panggilan untuk `kfree()`. Hati-hati manajemen memori—yang meliputi melepaskan memori untuk mencegah kebocoran memori—adalah penting ketika mengembangkan kernel-kode tingkat

## Bagian II Tugas

Dalam modul entry point, membuat linked list yang berisi lima struct ulang tahun elemen. Melintasi linked list dan output isinya ke kernel log buffer. Memanggil perintah `dmesg` untuk memastikan daftar ini dibangun dengan baik sekali kernel modul telah dimuat. Dalam modul titik keluar, menghapus elemen dari linked list dan kembali memori kembali ke kernel. Lagi, memanggil perintah `dmesg` untuk memeriksa bahwa daftar tersebut sudah dihapus setelah modul kernel yang telah dibongkar.

