SISTEME DE OPERARE MODERNE

Ediția a patra

Andrew S. Tanenbaum

Herbert Bos

Pretince Hall PTR
Upper Saddle River, New Jersey 07458
www.phptr.com

/

PREFAȚA

Ediția a patra a acestei cărți este foarte diferită de a treia. Întrucât dezvoltarea sistemelor de operare nu rămâne pe loc, aducerea materialului într-o stare modernă a necesitat un număr mare de mici modificări. Capitolul despre sistemele de operare multimedia a fost mutat pe Internet, în principal pentru a face loc materialelor noi și pentru a împiedica cartea să crească până la o dimensiune absolut incontrolabilă. Capitolul Windows Vista a fost șters deoarece Vista nu a reușit să se ridice la nivelul așteptărilor companiei Microsoft. La fel și capitolul Symbian, deoarece sistemul este acum mult mai puțin răspândit. Vista a fost înlocuit cu Windows 8, iar Symbian cu Android. În plus, a fost inclus un capitol nou despre virtualizare.

1. INTRODUCERE

Un computer modern este format dintr-unul sau mai multe procesoare, memoria operativă, memoria secundară, o imprimantă, o tastatură, un mouse, un display, echipamente de rețea și diverse dispozitive de intrare/ieșire. În consecință, este un sistem extrem de complex. Dacă un programator simplu ar fi obligat să înțeleagă complexitatea tuturor acestor dispozitive, atunci nu-i va mai rămâne timp pentru a scrie cod. Mai mult, gestionarea tuturor acestor componente și utilizarea lor optimă este o sarcină foarte dificilă. Din acest motiv, calculatoarele sunt echipate cu un software special numit **Sistem de Operare**, sarcina căruia este gestionarea programelor utilizatorului, precum și administrarea tuturor resurselor menționate anterior.

Majoritatea cititorilor au deja experiență cu sisteme de operare, cum ar fi Windows, Linux, Unix FreeBSD sau Mac OS X, dar aspectul lor poate varia. Programele folosite de utilizatori pentru a interacționa cu calculatorul, de obicei numite shell, atunci când se bazează pe utilizarea textului sau interfața grafică de utilizator (GUI), nu fac parte din sistemul de operare, deși folosesc acest sistem.

Schematic, principalele componente luate în considerare aici sunt prezentate în Fig. 1.1.

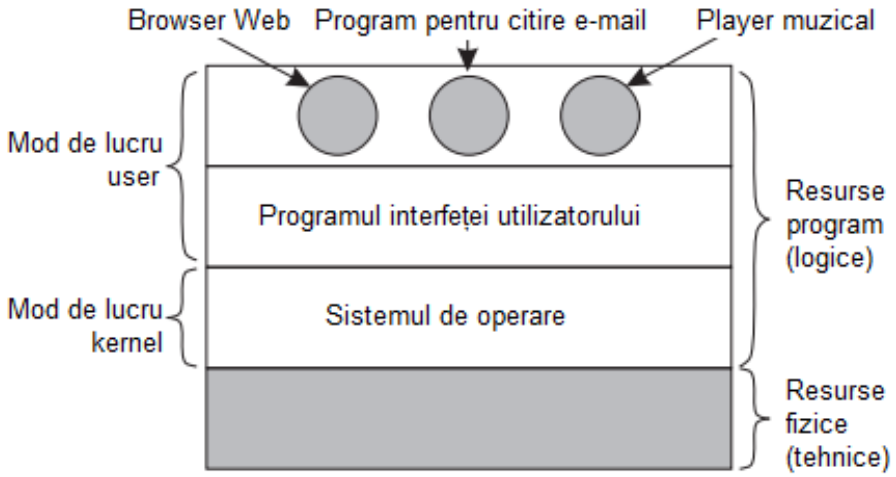

Fig. 1.1. Locul sistemului de operare în cadrul resurselor program

Partea inferioară a imaginii reprezintă resursele fizice (hardware). Include microcircuite, plăci, discuri, tastatură, monitor și alte componente fizice. Mai sus de hardware este software-ul. Majoritatea computerelor au două moduri de operare: modul kernel și modul utilizator. Sistemul

de operare este cea mai fundamentală piesă de software care funcționează în modul kernel (acest mod se mai numește și modul supervizor). În acest mod, sistemul are acces complet la toate componentele tehnice și poate folosi orice instrucțiune pe care calculatorul este capabil să o execute. Restul software-ului rulează în modul utilizator, în care este disponibil doar un subset de instrucțiuni ale mașinii. De exemplu, programelor care operează în modul utilizator sunt interzise să utilizeze instrucțiuni care controlează mașina sau să efectueze operațiuni de Intrare/Ieșire (I/O). Vom reveni de mai multe ori la diferențele dintre modurile kernel și utilizator. Aceste diferențe au o influență decisivă asupra sistemului de operare.

Programele de interfață ale utilizatorului - shell-ul sau interfața grafică - se află la cel mai inferior nivel de software executat în modul utilizator și permit utilizatorului să ruleze alte programe, cum ar fi un browser web, un cititor de e-mail sau un player de muzică. Aceste programe de asemenea folosesc în mod activ sistemul de operare.

Locul sistemului de operare este prezentat în Fig. 1.1. Acesta lucrează direct cu hardware-ul și reprezintă baza pentru restul software-ului.

O diferență importantă între sistemul de operare și software-ul obișnuit (care operează în modul utilizator) este următoarea: dacă utilizatorul nu este mulțumit de un anumit cititor de e-mail, poate alege un alt program sau, dacă dorește, își poate scrie propriul program, dar nu poate scrie propriul program de gestionare a întreruperilor ceasului de sistem, care face parte din sistemul de operare și este protejat la nivel hardware de orice încercare de modificare din partea utilizatorului. Această distincție este uneori mai puțin pronunțată în sistemele încorporate (care pot să nu dispună de modul kernel) sau în sistemele interpretate (cum ar fi sistemele bazate pe limbajul Java, în care pentru separarea componentelor nu sunt utilizate resursele tehnice ci un interpretor).

Multe sisteme au, de asemenea, programe care rulează în modul utilizator, dar care ajută sistemul de operare sau îndeplinească funcții speciale. De exemplu, programele care permit utilizatorilor să își schimbe parolele sunt destul de frecvente. Acestea nu fac parte din sistemul de operare și nu rulează în modul kernel, dar se înțelege că acestea îndeplinesc o funcție importantă și trebuie protejate în mod special. În unele sisteme, această idee este dusă la o formă extremă, iar acele zone care aparțineau în mod tradițional sistemului de operare (de exemplu, sistemul de fișiere) funcționează în spațiul utilizatorului. În aceste sisteme este dificil de trasat o graniță clară. Toate programele care rulează în modul kernel fac parte în mod clar din sistemul de operare, însă unele programe care rulează în afara acestui mod pot face parte din sistemul de operare sau cel puțin sunt strâns legate de el.

Sistemele de operare diferă de programele de utilizator (adică de aplicații) nu numai prin locația lor. Caracteristic pentru ele sunt volumul relativ mare al codului, structura complexă și durata de viață lungă. Codul sursă al unui sistem de operare precum Linux sau Windows constă din aproximativ 5 milioane de linii. Pentru a ne imagina acest volum, să tipărim mental 5 milioane de linii în format carte de 50 de linii pe pagină și 1000 de pagini în fiecare volum (care este mai mare decât această carte). Ar fi nevoie de 100 de volume pentru a tipări această cantitate de cod, ceea ce înseamnă practic un întreg raft de cărți. Vă puteți imagina dacă vi s-ar da sarcina să asigurați mentenanța unui astfel de sistem și în prima zi șeful dumneavoastră v-ar duce la un raft de cărți și v-ar spune: "Asta e tot ce trebuie să învețI." Iar aceasta este doar pentru partea care rulează în modul kernel. Dacă includem și bibliotecile comune necesare, Windows are peste 70 de milioane de linii de cod (tipărite pe hârtie, acestea ar ocupa 10-20 de rafturi), fără a mai pune la socoteală principalele programe de aplicații (cum ar fi Windows Explorer, Windows Media Player etc.).

Acest lucru explică de ce sistemele de operare trăiesc atât de mult timp - sunt foarte greu de creat și, odată ce a fost creat unul, proprietarul este reticent să îl arunce și să înceapă să creeze unul nou. Acesta este motivul pentru care sistemele de operare evoluează pe o perioadă lungă de timp. Familia Windows 95/98/Me a fost în esență un singur sistem de operare, iar familia Windows NT/2000/XP/Vista/Windows 7 a fost un altul. Pentru utilizator, acestea semănau unul cu celălalt, deoarece Microsoft s-a asigurat că interfața de utilizator la Windows 2000/XP/Vista/Windows 7 era foarte asemănătoare cu cea a sistemului pe care îl înlocuia, care, de cele mai multe ori, era Windows 98. Cu toate acestea, Microsoft a avut câteva motive destul de întemeiate pentru a renunța la Windows 98, pe care le vom revedea atunci când vom intra într-un studiu detaliat al sistemului Windows în Capitolul 11.

Un alt exemplu, folosit pe parcursul cărții, este sistemul de operare UNIX, variantele și clonele sale. Acesta a evoluat de-a lungul anilor, existând în versiuni bazate pe sistemul original, cum ar fi System V, Solaris și FreeBSD. Linux, pe de altă parte, are o nouă fundație software, deși modelul său se bazează mult pe UNIX și are un grad ridicat de compatibilitate cu acesta. Exemplele referitoare la UNIX vor fi folosite pe tot parcursul cărții, în timp ce Linux este tratat în detaliu în capitolul 10.

Acest capitol abordează pe scurt unele dintre aspectele cheie ale sistemelor de operare, învățând ce sunt acestea, examinând istoria lor și tot ceea ce s-a întâmplat în jurul lor, analizând câteva concepte de bază ale sistemelor de operare și structura acestora.

Multe subiecte importante vor fi abordate mai detaliat în capitolele următoare.

1.1 Ce este un sistem de operare?

Este dificil să se dea o definiție precisă a unui sistem de operare (SO). Se poate spune că este un software care rulează în modul kernel, dar nici măcar această afirmație nu va fi întotdeauna adevărată. O parte a problemei constă în faptul că sistemele de operare îndeplinesc două funcții semnificativ diferite: ele (1) **oferă** programatorilor de aplicații (și, bineînțeles, aplicațiilor) **un set de resurse abstracte** ușor de înțeles, care înlocuiesc setul neordonat de resurse fizice și (2) **gestionează** aceste resurse abstracte. În funcție de persoana care vorbește, este posibil să auzim mai mult despre prima sau despre a doua dintre aceste funcții. Nouă ne revine sarcina să analizăm ambele funcții.

1.1.1 Sistemul de operare ca mașină extinsă

Arhitectura majorității calculatoarelor (sistemul de instrucțiuni, organizarea memoriei, sistemul de intrare/ieșire și magistralele) la nivelul limbajului cod mașină este prea primitivă și incomodă pentru a fi utilizată în programe, în special în sistemele I/O. Pentru a pune discuția în context, să ne uităm la hard disk-urile moderne SATA (Serial ATA), care sunt utilizate în majoritatea computerelor. O carte publicată de Anderson Publishing în 2007, care descria interfața de disc pe care programatorii trebuiau să o învețe pentru a utiliza unitatea, conținea peste 450 de pagini. De atunci, interfața a fost revizuită de mai multe ori și a devenit chiar mai complexă decât era în 2007. Este clar că niciun programator sănătos la cap nu ar dori să se ocupe de o astfel de unitate la nivel hardware. În ajutorul programatorului, hardware-ul este gestionat de un software numit driver de disc, care oferă, fără alte precizări, o interfață pentru citirea și scrierea blocurilor de disc. SO conțin multe drivere pentru a controla dispozitivele de I/O.

Dar, pentru majoritatea aplicațiilor, chiar și acest nivel este prea scăzut. Din această cauză toate sistemele de operare oferă un alt nivel (mai înalt) de abstractizare pentru utilizarea discului - fișierele. Utilizând această abstractizare, programele pot crea, scrie și citi fișiere, fără a intra în detaliile legate de modul în care funcționează de fapt hardware-ul.

Această abstractizare este cheia pentru gestionarea complexității. O bună abstractizare transformă o sarcină aproape imposibilă în două sarcini ușor de gestionat. Prima dintre aceste sarcini constă în definirea și implementarea abstracțiilor, iar cea de-a doua este utilizarea acestor abstracții pentru a rezolva problema curentă. O abstractizare pe care aproape orice utilizator de calculator o poate înțelege este fișierul menționat anterior. Acesta reprezintă un set de informații utile, de exemplu o fotografie digitală, un mesaj de e-mail salvat sau o pagină web. Fotografiile, e-mailurile și paginile web sunt mult mai ușor de utilizat decât discurile SATA (sau alte componente de disc). Sarcina unui sistem de operare este de a crea o abstractizare bună, iar apoi de a implementa și gestiona obiectele abstracte create în cadrul acestei abstractizări. În această carte, abstracțiunile vor avea o prioritate foarte mare, deoarece ele reprezintă una dintre cheile pentru înțelegerea sistemelor de operare.

Având în vedere importanța acestui concept, merită să îl exprimăm în cuvinte ușor diferite. Cu tot respectul pentru proiectanții Macintosh, trebuie remarcat faptul că hardware-ul nu se evidențiază în mod deosebit prin finețe sau eleganță. Procesoarele, blocurile de memorie, discurile și alte componente din lumea reală sunt mult prea complexe, oferind interfețe dificile, incomode, incoerente și inconsistente pentru cei care trebuie să scrie cod pentru ele. Uneori, acest lucru se datorează necesității de a susține compatibilitatea retroactivă, alteori dorinței de a economisi bani, iar uneori, dezvoltatorii de hardware pur și simplu nu realizează (sau nu vor să realizeze) cât de multe probleme le creează dezvoltatorilor de software. Unul dintre obiectivele principale ale unui sistem de operare este de a ascunde hardware-ul și software-ul existent (și dezvoltatorii acestuia) sub abstracțiuni frumoase, elegante și neschimbătoare, care au fost create pentru a le înlocui și a funcționa corect. Sistemele de operare transformă urâțenia în frumusețe (figura 1.2).

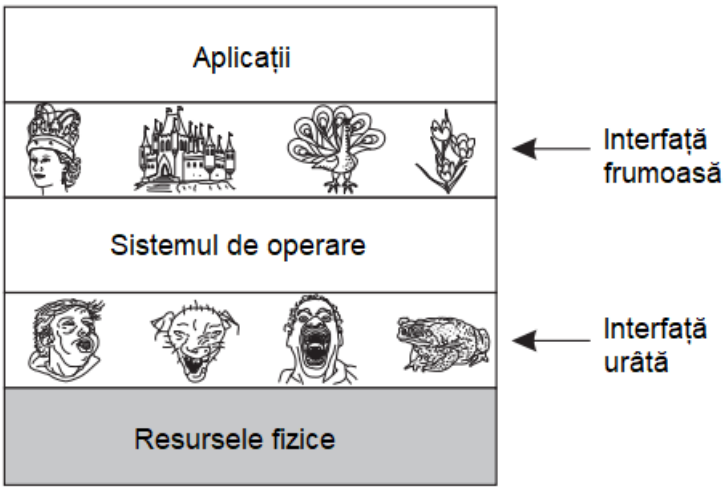

Fig. 1.2. Sistemul de operare transformă resursele tehnice urâte în abstracții frumoase

Trebuie remarcat faptul că adevărații "beneficiari" ai sistemelor de operare sunt aplicațiile (desigur, nu fără ajutorul programatorilor de aplicații). Anume acestea sunt cei care lucrează direct cu sistemul de operare și cu abstracțiunile acestuia. Iar utilizatorii finali lucrează cu abstracțiunile oferite de interfața utilizatorului - fie o linie de comandă shell, fie o interfață grafică. Abstracțiunile interfeței utilizatorului pot fi similare abstracțiunilor furnizate de sistemul de operare, dar nu întotdeauna este cazul. Pentru a clarifica acest aspect, luați în considerare desktop-ul normal al SO Windows și linia de comandă. Ambele sunt programe care rulează pe sistemul de operare Windows și utilizează abstracțiile furnizate de acest sistem, dar oferă interfețe de utilizator semnificativ diferite. În mod similar, utilizatorii de Linux care rulează în

Gnome sau KDE văd o imagine total diferită decât utilizatorii de Linux care rulează pe X Window System, însă abstracțiunile care stau la baza SO sunt aceleași în ambele cazuri.

În această carte vom studia în detaliu abstracțiunile disponibile pentru aplicații, dar nu ne vom aprofunda prea mult în interfețele cu utilizatorul. Acesta este un subiect complex și important, dar care are prea puțin de-a face cu sistemele de operare.

1.1.2 Sistemul de operare ca administrator de resurse

Punctul de vedere conform căruia sistemul de operare oferă în principal abstracțiuni pentru programele de aplicații este o viziune de sus în jos (top down). Susținătorii punctului de vedere alternativ, de jos în sus (down-top), consideră că sistemul de operare există pentru a controla toate părțile unui sistem complex. Calculatoarele moderne sunt compuse din procesoare, memorie, cronometre, discuri, mouse-uri, interfețe de rețea, imprimante și o gamă largă de alte dispozitive. Susținătorii viziunii de jos în sus consideră că sarcina sistemului de operare este de a asigura o alocare ordonată și controlată a procesoarelor, memoriei și dispozitivelor de I/O între diversele programe care pretind să le utilizeze.

Sistemele de operare moderne permit rularea simultană a mai multor programe. Imaginați-vă ce s-ar întâmpla dacă toate cele trei programe care rulează pe același computer ar încerca să tipărească simultan pe aceeași imprimantă. Primele câteva rânduri ale imprimării pot fi din programul nr. 1, următoarele câteva rânduri din programul nr. 2, apoi câteva rânduri din programul nr. 3 și așa mai departe. Rezultatul va fi un haos total. Sistemul de operare este conceput pentru a face ordine în haosul potențial prin stocarea pe disc a tuturor ieșirilor destinate imprimantei. Sistemul de operare dispune de o memorie tampon pe disc pentru toate datele de ieșire destinate imprimantei. Atunci când un program își termină activitatea sistemul de operare poate direcționa către imprimantă fișierul de pe disc în care au fost stocate datele programului și, în același timp, un alt program poate continua să genereze date către imprimantă fără să-și dea seama că ieșirea nu ajunge de fapt (pentru moment) la imprimantă.

Atunci când mai mulți utilizatori lucrează cu un computer (sau o rețea) nevoile de gestionare și securizare a memoriei, intrărilor/ieșirilor și a altor resurse cresc dramatic. În caz contrar, utilizatorii vor interfera cu munca celorlalți. În plus, utilizatorii sunt adesea nevoiți să partajeze nu numai hardware-ul, dar și informațiile (fișiere, baze de date etc.). Pe scurt, susținătorii acestei viziuni asupre sistemului de operare consideră că sarcina principală a acestuia este de a ține evidența care program utilizează o resursă, pentru a satisface solicitările de resurse, fiind responsabil pentru utilizarea acestora și de a lua decizii cu privire la solicitările contradictorii din partea diferitor programe și utilizatori diferiți.

Gestionarea resurselor implică multiplexarea (alocarea) resurselor în două moduri diferite: în timp și în spațiu. Atunci când o resursă este partajată în timp, diferite programe sau utilizatori o folosesc pe rând: mai întâi unul, apoi altul și așa mai departe. De exemplu, în cazul în care există un singur procesor și mai multe programe care încearcă să ruleze pe el, sistemul de operare alocă mai întâi procesorul unui program. După ce acesta a fost rulat suficient de mult timp, procesorul este alocat unui alt program, apoi unui al treilea program și tot așa. În cele din urmă, primul program poate primi din nou procesorul. Stabilirea exactă a modului în care resursa va fi alocată în timp - cine va fi următorul consumator și pentru cât timp - este sarcina sistemului de operare. Partajarea imprimantei reprezintă alt exemplu de multiplexare în timp. Atunci când în coada de așteptare a unei imprimante există mai multe lucrări de imprimare, trebuie să se decidă care va fi următoarea lucrare de tipărire.

Un alt tip de partajare a resurselor este partajarea spațială. În loc să alterneze, fiecare client primește o parte din resursa partajată. De exemplu, memoria RAM este, de obicei, partajată

între mai multe programe în curs de execuție, astfel încât acestea să poată fi toate în memorie în orice moment (de exemplu, utilizând pe rând procesorul). Cu condiția să existe suficientă memorie pentru mai multe programe, este mai eficient să se aloce memoria mai multor programe simultan, decât să fie alocată toată memoria unui singur program, mai ales dacă acesta are nevoie doar de o mică parte din spațiul total. Desigur, în acest caz pot apărea probleme de partajare echitabilă, de securitate și așa mai departe, care trebuie rezolvate de sistemul de operare. O altă resursă partajată spațial este discul rigid. Mai multe sisteme permit stocarea fișierelor care aparțin diferitor utilizatori pe aceeași unitate în același timp. Alocarea spațiului pe disc și ținerea evidenței cine folosește care blocuri de disc este o sarcină tipică a unui sistem de operare.

1.2 Istoria sistemelor de operare

Istoria sistemelor de operare datează de mulți ani. În următoarele compartimente ale acestei cărți vom discuta pe scurt câteva dintre principalele momente ale acestei evoluții. Deoarece sistemele de operare au apărut și au evoluat în procesul de creare a calculatoarelor, aceste evenimente sunt strâns legate din punct de vedere istoric. Din acest motiv, pentru a obține un tablou al dezvoltării sistemelor de operare, vom examina generațiile succesive de calculatoare. O atare schemă a relațiilor dintre generațiile de sisteme de operare și de calculatoare este foarte aproximativă, dar oferă o anumită structură fără de care ar fi complicat de înțeles ceva.

Primul calculator digital adevărat a fost inventat de matematicianul englez Charles Babbage (1792-1871). Deși Babbage și-a petrecut cea mai mare parte a vieții construindu-și mașina analitică, acesta nu a reușit niciodată să o facă să funcționeze. Fiind o mașină pur mecanică, tehnologia din acea vreme nu era suficient de avansată pentru a realiza multe dintre piesele și mecanismele de mare precizie. Inutil să spunem că mașina sa nu avea un sistem de operare.

Este un fapt istoric interesant că Babbage și-a dat seama că pentru o mașină analitică avea nevoie de software, așa că a angajat o tânără pe nume Ada Lovelace, fiica celebrului poet britanic George Byron. Dânsă a devenit primul programator din lume, iar limbajul de programare Ada®, creat în zilele noastre, a fost numit în cinstea ei.

1.2.1 Prima generatie (1945 - 1955): tuburi cu vid

După eforturile nereușite ale lui Babbage, s-au înregistrat puține progrese în proiectarea calculatoarelor digitale până în timpul celui de-al Doilea Război Mondial, care a stimulat o explozie de lucrări în acest sens. Profesorul John Atanasoff și doctorandul său Clifford Berry au creat ceea ce este considerat în prezent primul calculator digital funcțional la Universitatea din Iowa. Acesta folosea 300 de tuburi electronice. Cam în aceeași perioadă, Konrad Zuse din Berlin a construit computerul Z3 bazat pe relee electromecanice. În 1944, un grup de oameni de știință (inclusiv Alan Turing) de la Bletchley Park din Marea Britanie a construit și programat “Colossus”, la Harvard Howard Aiken a construit ”Mark 1”, iar la Universitatea din Pennsylvania William Mauchley și doctorandul său John Presper Eckert au construit ”Eniac”. Unele dintre aceste mașini erau digitale, altele foloseau tuburi electronice, erau programabile dar destul de primitive și necesitau multe secunde pentru a produce chiar și cele mai simple calcule.

La începuturile erei calculatoarelor, fiecare mașină era proiectată, construită, programată, operată și întreținută de același grup de persoane (de obicei ingineri). Toate programările se făceau exclusiv în limbajul mașinilor sau, chiar mai rău, prin asamblarea circuitelor electrice, iar mii de fire trebuiau conectate la panouri de conexiuni pentru a controla funcțiile de bază ale mașinilor. Limbajele de programare (chiar și cele de asamblare) nu erau încă cunoscute. Nimeni nu auzise niciodată de sisteme de operare. Modul de operare al programatorului era să

se înscrie pentru un anumit timp de lucru într-un registru special, apoi, cînd îi revine timpul, să coboare în camera de control, să-și conecteze panoul de patch-uri la computer și să petreacă următoarele câteva ore, sperând că niciunul dintre cele aproximativ 20 000 de tuburi electronice nu vor ieși din funcție în timpul procesului. În esență, toate sarcinile care trebuiau îndeplinite se reduceau la calcule matematice și numerice simple, cum ar fi precizrea tabelelor de sinusuri, cosinusuri și logaritmi sau calcularea traiectoriilor proiectilelor de artilerie.

Când s-a propus folosirea cartelelor perforate, la începutul anilor 1950, situația s-a îmbunătățit oarecum. În loc să se folosească panouri de comutare, acum programele puteau fi scrise pe cartele și citite de pe acestea, dar în rest procedura de operare a rămas neschimbată.

1.2.2 A doua generație (1955 - 1965): tanzistoarele și tratarea pe loturi

Inventarea tranzistoarelor în anul 1948 și introducerea lor în locul tuburilor cu vid (mijlocul anilor 1950) a schimbat radical situația. Calculatoarele au devenit suficient de fiabile pentru a putea fi fabricate ân scopuri comerciale, cu speranța că vor continua să funcționeze suficient de mult timp pentru a face o muncă utilă. Acuma se produce prima separare între proiectanți, constructori, operatori, programatori și personalul de întreținere.

Aceste mașini, numite acum mainframe-uri, erau închise în săli mari speciale de calculatoare, cu aer condiționat, cu operatori profesioniști care să le gestioneze. Numai marile corporații, marile agenții guvernamentale sau universități își puteau permite să plătească prețul de mai multe milioane de dolari. Pentru a rula o lucrare (adică un program sau un set de programe), un programator scria mai întâi programul pe hârtie (în FORTRAN sau în limbaj de asamblare), îl transpunea pe cartele perforate, aducea pachetul de cartele în camera de introducere a datelor, îl înmâna unuia dintre operatori și se ducea să bea o cafea până când rezultatul era gata.

Când execuția unei lucrări era încheiată, un operator se ducea la imprimantă, selecta listingul și îl ducea în camera de ieșire, astfel încât programatorul să-l poată prelua mai târziu. Apoi, operatorul lua unul dintre pachetele de cartele din camera de intrare și îl introducea. Dacă era nevoie de compilatorul FORTRAN, operatorul trebuia să îl ia dintr-un dulap de fișiere și să îl încarce în memoria calculatorului. Se pierdea mult timp cu calculatorul cu această plimbare a operatorilor prin sala de mașini.

Având în vedere costul ridicat al echipamentului, nu este surprinzător faptul că oamenii au căutat rapid modalități de a reduce timpul pierdut. Soluția adoptată a fost tratarea pe loturi. Ideea era de a colecta o cutie plină de lucrări în camera de intrare și apoi de a le stoca pe o bandă magnetică folosind un calculator mic (relativ) ieftin, cum ar fi IBM 1401, care era destul de bun la citirea cartelelor, la copierea benzilor și la imprimarea rezultatelor, dar deloc bun la calcule numerice. Alte mașini mult mai scumpe, cum ar fi IBM 7094, erau fost folosite pentru calculul real. Această situație este prezentată în fig. 1-3.

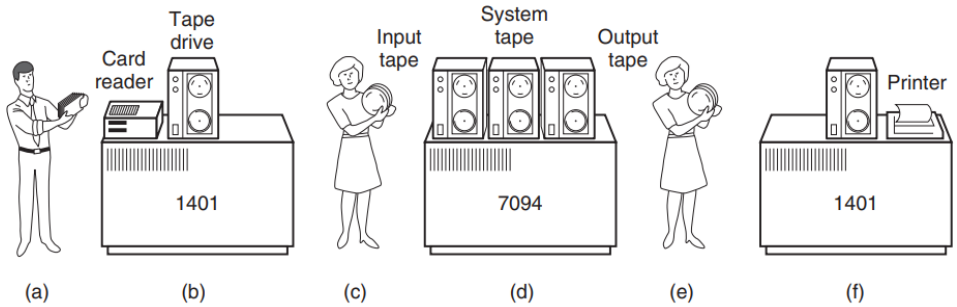

Fig. 1-3. Tratarea pe loturi. (a) Programatorii aduc cartelele perforate la 1401. (b) 1401 citește lotul de lucrări pe bandă. (c) Operatorul duce banda de intrare la 7094. (d) 7094 efectuează calculul. (e) Operatorul duce banda de ieșire la 1401. (f) 1401 tipărește ieșirea.

După aproximativ o oră de colectare a unui lot de lucrări, cardurile erau citite pe o bandă magnetică, care era transportată în sala mașinilor, unde era montată pe o unitate de bandă. Operatorul încărca apoi un program special (strămoșul sistemului de operare de astăzi), care citea prima lucrare de pe bandă și o executa. Rezultatul era scris pe o a doua bandă, în loc să fie tipărit. După ce o lucrare se termina, sistemul de operare citea automat următoarea lucrare de pe bandă și începea să o ruleze. Când întregul lot era gata, operatorul scotea benzile de intrare și de ieșire, înlocuia banda de intrare cu următorul lot și aducea banda de ieșire la un 1401 pentru imprimare off line (adică fără a fi conectată la calculatorul principal).

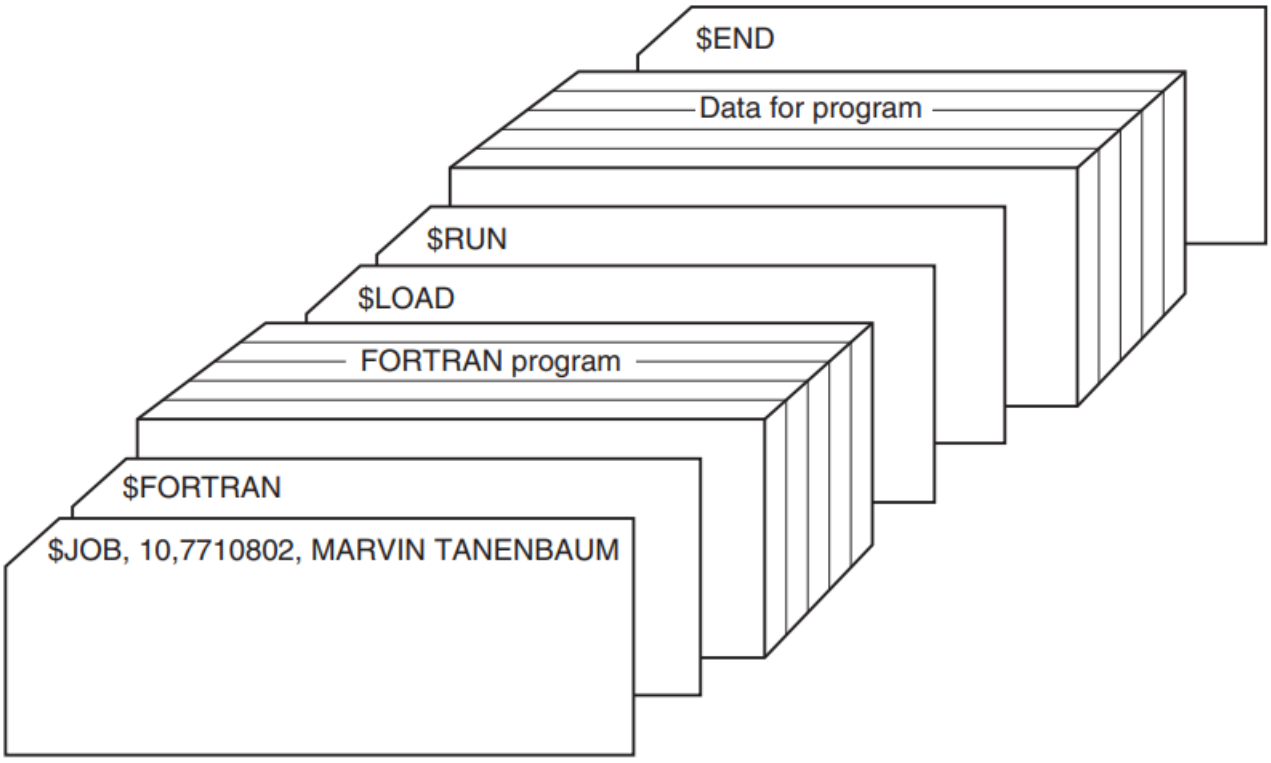

Fig. 1.4. Structura unei lucrări în Fortran Monitor System (FMS)

Structura unei sarcini de intrare (lucrări, job) tipice este prezentată în figura 1-4. Aceasta începe cu o cartelă **`$JOB`**, specificând timpul maxim de execuție în minute, numărul de cont care trebuie debitat și numele programatorului. Apoi venea o cartelă **`$FORTRAN`**, indicând sistemului de operare să încarce compilatorul FORTRAN de pe banda de sistem. Aceasta era urmată direct de programul care urma să fie compilat și apoi de o cartelă **`$LOAD`**, care indica sistemului de operare să încarce programul obiect tocmai compilat. (Programele compilate erau deseori scrise pe o bandă scratch[1] și trebuiau să fie încărcate în mod explicit). Apoi venea cardul **`$RUN`**, care îi spunea sistemului de operare să ruleze programul cu datele care îl urmau. În cele din urmă, cardul **`$END`** marca sfârșitul lucrării. Aceste carduri de control primitive au fost precursorii shell-urilor moderne și ai interpretorilor de linii de comandă.

[1] O bandă de scratch este o bandă magnetică care este utilizată pentru stocare temporară și care poate fi reutilizată sau ștearsă după finalizarea unei lucrări sau a unei execuții de procesare.

Computerele mari din a doua generație au fost utilizate în principal pentru calcule științifice și inginerești, cum ar fi rezolvarea ecuațiilor diferențiale parțiale care apar adesea în fizică și inginerie. Acestea erau în mare parte programate în FORTRAN și/sau în limbaj de asamblare. Sistemele de operare tipice erau FMS (Fortran Monitor System) și IBSYS, sistemul de operare al IBM pentru 7094.

1.2.3 Generatia a treia (1965 - 1980): Circuite integrate si multiprogramare

La începutul anilor 1960 majoritatea producătorilor de calculatoare aveau două linii de produse distincte și incompatibile. Pe de o parte, existau computerele științifice de mari dimensiuni, orientate pe cuvinte, cum ar fi IBM 7094, care erau utilizate pentru calcule numerice de putere industrială în știință și inginerie. Pe de altă parte, erau computerele comerciale, orientate spre caractere, cum ar fi IBM 1401, utilizate pe scară largă pentru sortare și imprimare de către bănci și companii de asigurări.

Dezvoltarea și întreținerea a două linii de produse complet diferite era o soluție costisitoare pentru producători. În plus, mulți clienți noi la început aveau nevoie de mașini mici, dar mai târziu doreau o mașină mai mare, care să le ruleze toate programele vechi, dar mai rapid. IBM a încercat să rezolve aceste două probleme dintr-o singură lovitură prin introducerea mașinilor System/360. Linia 360 era o serie de mașini compatibile cu software-ul de la modele 1401 până la unele mult mai mari, mai performante decât puternica 7094. Mașinile se deosebeau doar prin preț și performanță (capacitatea memoriei, viteza procesorului, numărul de dispozitive I/O permise și așa mai departe). Deoarece toate aveau aceeași arhitectură și același set de instrucțiuni, programele scrise pentru o mașină puteau rula pe toate celelalte - cel puțin în teorie. (Dar, așa cum se zice că a spus Yogi Berra: "În teorie, teoria și practica sunt la fel; în practică, nu sunt.") Deoarece System/360 a fost proiectat pentru a gestiona atât calculele științifice (numerice), cât și cele comerciale, o singură familie de mașini putea satisface nevoile tuturor clienților. În anii următori, IBM a scos pe piață succesori compatibili cu linia 360, folosind o tehnologie mai modernă, cunoscută sub numele de 370, 4300, 3080 și 3090. ZSeries este cel mai recent descendent al acestei linii, deși s-a îndepărtat considerabil de original.

IBM 360 a fost prima linie majoră de calculatoare care a utilizat circuite integrate (de mici dimensiuni), oferind astfel un avantaj considerabil de preț/performanță față de mașinile din a doua generație, care erau construite din tranzistori individuali. A fost un succes imediat, iar ideea unei familii de computere compatibile a fost adoptată în scurt timp de toți ceilalți mari producători. Descendentele acestor mașini sunt utilizate și astăzi în centrele de calcul. În prezent, ele sunt adesea folosite pentru gestionarea unor baze de date uriașe (de exemplu, pentru sistemele de rezervare a companiilor aeriene) sau ca servere pentru site-urile World Wide Web care trebuie să proceseze mii de cereri pe secundă.

Cel mai mare avantaj al ideii ''familiei unice'' a reprezentat în același timp și cea mai mare slăbiciune a acesteia. Intenția inițială era ca toate programele, inclusiv sistemul de operare, OS/360, să funcționeze pe toate modelele. Sistemul de operare trebuia să funcționeze pe sisteme mici, care adesea înlocuiau doar 1401 pentru copierea cartelelor perforate pe bandă și pe sisteme foarte mari, care înlocuiau 7094 pentru a face prognoze meteo și alte calcule grele, să funcționeze atât pe sisteme cu puține periferice, cât și pe sisteme cu multe periferice, să funcționeze în medii comerciale și în medii științifice. Plus la toate, trebuia să fie eficiente pentru toate aceste utilizări diferite.

IBM (sau oricine altcineva, de altfel) nu avea cum să scrie un software care să îndeplinească toate aceste cerințe contradictorii. Rezultatul a fost un sistem de operare enorm și extraordinar de complex, probabil cu două-trei ordine de mărime mai mare decât FMS. Acesta era format

din milioane de linii de cod scrise de mii de programatori și conținea mii și mii de erori, care necesitau un flux continuu de noi versiuni în încercarea de a le corecta. Fiecare nouă versiune corecta unele erori și introducea altele noi, astfel încât numărul de erori rămânea probabil constant în timp.

Unul dintre proiectanții OS/360, Fred Brooks, a scris ulterior o carte spirituală și incisivă (Brooks, 1995) în care descrie experiența proprie cu OS/360. Deși ar fi imposibil să rezumăm cartea aici, este suficient să spunem că pe copertă apare o turmă de animale preistorice blocate într-o groapă cu smoală. Coperta cărții lui Silberschatz et al. (2012) face o remarcă similară cu privire la faptul că sistemele de operare sunt dinozauri.

În ciuda dimensiunii și a problemelor sale enorme, OS/360 și sistemele de operare similare din a treia generație produse de alți producători de calculatoare satisfăceau de fapt destul de bine majoritatea cerințelor clienților. Mai mult, aceste sisteme de operare au făcut populare mai multe tehnici-cheie absente în sistemele de operare din generația a doua. Probabil cea mai importantă dintre acestea a fost multiprogramarea. Pe mașina 7094, atunci când lucrarea curentă se întrerupea pentru a aștepta finalizarea intrării de pe o bandă sau a unei alte operațiuni de intrare/ieșire, unitatea centrală de procesare stătea pur și simplu inactivă până la terminarea operațiunii de intrare/ieșire. În cazul calculelor științifice puternic legate de procesor, I/O nu sunt frecvente, astfel încât acest timp pierdut nu este semnificativ. Însă, la prelucrarea datelor comerciale, timpul de așteptare pentru I/O poate reprezenta adesea 80 sau 90% din timpul total, așa că trebuia să se facă ceva pentru a evita ca procesorul (costisitor) să fie atât de mult timp inactiv.

Soluția acestei probleme a fost de a împărți memoria în mai multe părți,numite partiții, cu o sarcină diferită în fiecare partiție, așa cum este prezentat în figura 1-5.

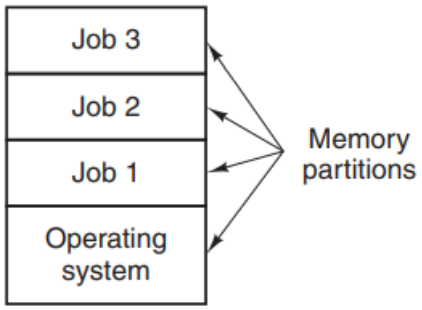

Fig. 1.5. Sistem multitasking cu trei lucrări în memorie

În timp ce o lucrare aștepta finalizarea I/O, o altă lucrare putea utiliza procesorul. Dacă în memoria principală se puteau afla simultan suficiente lucrări, procesorul central putea fi ținut ocupat aproape 100% din timp. Pentru a avea simultan mai multe lucrări în memorie erau necesare mecanisme hardware speciale care să asigure protecția reciprocă între lucrări. Calculatoarele din seria 360 și alte sisteme din generația a treia erau echipate cu astfel de mecanisme.

O altă caracteristică importantă a sistemelor de operare din generația a treia era capacitatea de a citi lucrările de pe cartelele perforate pe disc imediat ce erau aduse în sala de calculatoare. Apoi, ori de câte ori o lucrare în curs de execuție se termina, sistemul de operare putea încărca o nouă lucrare de pe disc în partiția acum goală și o putea executa. Această tehnică se numește spooling (de la Simultaneous Peripheral Operation On Line) și era utilizată și pentru ieșire. Cu introducerea spooling-ului, nu mai era nevoie de mașina 1401 și a dispărut o mare parte din lucrul cu benzile.

Deși sistemele de operare din generația a treia erau potrivite pentru calcule științifice de mare anvergură și pentru execuții masive de procesare a datelor comerciale, acestea erau încă sisteme de tip batch. Mulți programatori tânjeau după zilele primei generații, când aveau mașina numai pentru ei timp de câteva ore, pentru a-și putea depana rapid programele. În cazul SO din generația a treia, timpul dintre trimiterea unei sarcini și obținerea rezultatelor era adesea de câteva ore, astfel încât o singură virgulă greșit plasată putea face ca o compilare să eșueze, iar programatorul să piardă jumătate de zi. Programatorilor nu le plăcea acest lucru.

Această dorință de a avea un răspuns rapid a deschis calea pentru timesharing, o variantă de multiprogramare, în care fiecare utilizator are un terminal online. Într-un sistem de timesharing, dacă 20 de utilizatori sunt conectați și 17 dintre ei gândesc, vorbesc sau beau cafea, unitatea centrală poate fi alocată succesiv celor trei joburi care doresc să fie deservite. Deoarece oamenii care depanează programe emit de obicei comenzi scurte (de exemplu, compilați o procedură de cinci pagini[2]) mai degrabă decât comenzi lungi (de exemplu, sortați un fișier cu un milion de înregistrări), calculatorul poate oferi servicii rapide și interactive unui număr de utilizatori și, poate, de asemenea, să lucreze la sarcini mari de lucru pe loturi în fundal, atunci când procesorul este liber. Primul sistem de uz general cu partajarea timpului, CTSS (Compatible Time Sharing System), a fost dezvoltat la M.I.T. pe un 7094 special modificat (Corbato´ și al., 1962). Totuși timesharing-ul a devenit cu adevărat popular doar atunci când hardware-ul de protecție reciprocă necesar a fost implementat pe larg.

După succesul sistemului CTSS, M.I.T., Bell Labs și General Electric (la acea vreme un important producător de calculatoare) au decis să se lanseze în dezvoltarea unui "calculator utilitar", adică a unei mașini care să suporte câteva sute de utilizatori simultani. Ei s-au inspirat din sistemul energetic - atunci când ai nevoie de energie electrică, nu trebuie decât să introduci ștekerul în priză și, în limitele rezonabile, va exista atâta energie cât ai nevoie. Proiectanții acestui sistem, cunoscut sub numele de MULTICS (MULTiplexed Information and Computing Service), au avut în vedere o mașină uriașă care să ofere putere de calcul pentru toți locuitorii din zona Boston. Ideea că mașinile de 10.000 de ori mai rapide decât mainframe-ul lor GE-645 vor fi vândute (pentru mult sub 1.000 de dolari) cu milioanele era pe atunci din domeniul science fiction, și va fi implementată doar 40 de ani mai târziu. Foarte asemănătoare cu ideea trenurilor supersonice subacvatice transatlantice.

MULTICS a avut un succes discutabil. A fost conceput pentru a susține sute de utilizatori pe o mașină doar puțin mai puternică decât un PC cu Intel 386, deși avea o capacitate de I/O mult mai mare. Era o idee nu chiar trăsnită la vremea respectivă, deoarece oamenii de atunci erau capabili să creeze programe mici și eficiente, adică aveau o abilitate care s-a pierdut între timp. Au existat multe motive pentru care MULTICS nu a cucerit lumea, printre care nu ultimul fiind acela că a fost scris în limbajul de programare PL/I compilator pentru care a apărut doar peste câțiva ani, iar prima lui versiune poate fi numită funcțională doar cu mari rezerve. În plus, MULTICS a fost extrem de ambițios pentru epoca sa, la fel ca mașina analitică a lui Charles Babbage în secolul al XIX-lea.

MULTICS a introdus multe idei fundamentale în informatică, dar transformarea sa într-un produs serios și într-un succes comercial major s-a dovedit a fi mult mai dificilă decât era de așteptat. Bell Labs a renunțat la proiect, iar General Electric a renunțat cu totul la afacerea cu calculatoare. Cu toate acestea, M.I.T. a persistat și a reușit să facă MULTICS să funcționeze. În cele din urmă acesta a fost vândut ca produs comercial unei companii (Honeywell), care a cumpărat afacerea cu calculatoarele de la General Electric și a instalat aproximativ 80 de copii

[2] În această carte vom folosi termenii “procedură”, “subrutină” și “funcție” având același sens.

pe calculatoarele unor companii și universități importante din întreaga lume. Deși numărul de utilizatori MULTICS era mic, aceștia au dat dovadă de un angajament ferm față de sistem. De exemplu, General Motors, Ford și Agenția Națională de Securitate a SUA au renunțat la MULTICS abia la sfârșitul anilor '90, la 30 de ani de la lansare și după mai mulți ani în care au încercat să convingă Honeywell să actualizeze hardware-ul.

Spre sfârșitul secolului XX, ideea de a crea un astfel de calculator utilitar s-a stins, dar s-ar putea să revină sub forma cloud computing, în care computere relativ mici (inclusiv smartphone-uri, tablete și altele asemenea) sunt conectate la servere din centre de date vaste și îndepărtate, unde se realizează toate calculele, computerul local ocupându-se doar de interfața cu utilizatorul. Motivația în acest caz este că majoritatea oamenilor nu doresc să administreze sisteme informatice tot mai complexe și mai pretențioase și preferă ca această muncă să fie efectuată de o echipă de profesioniști, de exemplu, persoane care lucrează pentru compania care administrează centrul de date. Comerțul electronic evoluează deja în această direcție, diverse companii rulează e-mailuri pe servere multiprocesor la care se conectează mașini client simple, în spiritul proiectului MULTICS.

În ciuda lipsei de succes comercial, MULTICS a avut o influență uriașă asupra sistemelor de operare ulterioare (în special UNIX și derivatele sale, FreeBSD, Linux, iOS și Android). Acesta este descris în mai multe lucrări și într-o carte (Corbato´ et al., 1972; Corbato´ și Vyssotsky, 1965; Daley și Dennis, 1968; Organick, 1972; și Saltzer, 1974). Are, de asemenea, un site web activ, aflat la adresa www.multicians.org, cu informații despre sistem, proiectanți și utilizatori.

O altă evoluție majoră în timpul celei de-a treia generații a fost creșterea fenomenală a minicalculatoarelor, începând cu DEC PDP-1 în 1961. PDP-1 avea doar 4K de cuvinte pe 18 biți, dar la 120.000 de dolari pentru o mașină (mai puțin de 5% din prețul unui 7094), s-a vândut ca pâinea caldă. Pentru anumite tipuri de lucrări nenumerice, era aproape la fel de rapid ca 7094 și a dat naștere unei noi industrii. A fost urmat rapid de o serie de alte PDP-uri (spre deosebire de familia IBM, toate incompatibile) culminând cu PDP-11.

Unul dintre informaticienii de la Bell Labs care lucrase la proiectul MULTICS, Ken Thompson, a găsit ulterior un mic minicalculator PDP-7 pe care nu-l folosea nimeni și a început să scrie o versiune simplificată, pentru un singur utilizator, a sistemului MULTICS. Această lucrare s-a transformat mai târziu în sistemul de operare UNIX, care a devenit popular în lumea academică, în agențiile guvernamentale și în multe companii. Istoria dezvoltării SO UNIX a fost în repetate rânduri povestită în multe cărți (de exemplu, Salus, 1994). O parte din această poveste va fi prezentată în Cap. 10. Deocamdată, este suficient să spunem că, deoarece codul sursă era pe larg disponibil, diverse organizații și-au dezvoltat propriile versiuni (incompatibile), ceea ce a dus la un haos total. Au fost dezvoltate două versiuni de bază, System V, de la AT&T, și BSD (Berkeley Software Distribution) de la Universitatea California din Berkeley. Acestea au avut și unele variante minore.

Pentru a face posibilă scrierea de programe care să ruleze pe orice sistem UNIX, Institutul Inginerilor Electrotehniști și Electroniști (IEEE) a dezvoltat un standard pentru UNIX, numit POSIX, pe care majoritatea versiunilor UNIX îl acceptă în prezent. POSIX definește o interfață minimă pentru apelurile de sistem pe care sistemele UNIX compatibile trebuie să o suporte. De fapt, în prezent și alte sisteme de operare suportă interfața POSIX.

Ca o paranteză, merită menționat faptul că, în 1987, autorul acestei cărți a lansat o mică clonă de UNIX, numită MINIX, în scopuri educaționale. Din punct de vedere funcțional, MINIX este foarte asemănător cu UNIX, inclusiv respectă POSIX. De atunci, versiunea originală a evoluat în MINIX 3, care este foarte modular și are o fiabilitate foarte ridicată. Acesta are capacitatea de

a detecta și înlocui din mers modulele defecte sau chiar prăbușite (cum ar fi driverele de dispozitive I/O) fără repornire și fără a perturba programele în execuție. De asemenea, este disponibilă o carte care descrie funcționarea sa internă și listează codul sursă într-o anexă (Tanenbaum și Woodhull, 2006). Sistemul MINIX 3 este disponibil gratuit (inclusiv codul sursă) la adresa www.minix3.org.

Dorința de a avea o versiune industrială gratuită (spre deosebire de cea educațională) a sistemului MINIX l-a determinat pe un student finlandez, Linus Torvalds, să scrie **Linux**. Acest sistem a fost inspirat direct din MINIX și dezvoltat pe baza acestuia, suportând inițial diverse caracteristici MINIX (de exemplu, sistemul de fișiere MINIX). De atunci, a fost extins în multe feluri de către multe persoane, dar a păstrat în continuare o anumită structură de bază comună cu MINIX și cu UNIX. Cititorii interesați de o istorie detaliată a SO Linux și a mișcării Open Source pot citi cartea lui Glyn Moody (2001). Cea mai mare parte din ceea ce se va spune despre UNIX aici se aplică la System V, MINIX, Linux și alte versiuni și clone de UNIX.

1.2.4 Generatia a patra (1980 - până în prezent). Calculatoare personale

Următoarea perioadă în evoluția sistemelor de operare este asociată apariției circuitelor integrate mari (LSI, Large Scale Integration- Integrare pe scară largă) - cipuri de siliciu care conțin mii de tranzistori pe centimetru pătrat. Din punct de vedere al arhitecturii, calculatoarele personale (denumite inițial microcalculatoare) semănau foarte mult cu minicalculatoarele din clasa PDP-11, dar, bineînțeles, cu un preț diferit. În timp ce apariția minicalculatoarelor a permis departamentelor de companii și universități să dețină un calculator, apariția microprocesoarelor a oferit tuturor posibilitatea de a cumpăra un calculator personal.

În 1974, la lansarea microprocesorului Intel 8080, prima unitate centrală de procesare pe 8 biți de uz general, Intel avea nevoie de un sistem de operare care să poată fi folosit pentru a-l testa. Intel l-a angajat pe unul dintre consultanții săi, Gary Kildall, pentru a proiecta și a scrie sistemul de operare necesar. Kildall cu un prieten au proiectat mai întâi un controler pentru o unitate de dischetă de 8 inchi recent lansată de Shugart Associates și au atașat unitatea la un procesor Intel 8080. Astfel s-a născut primul microcalculator cu unitate de disc. Kildall a creat apoi un sistem de operare pe disc numit CP/M (Control Program for Microcomputers). Atunci când Kildall și-a revendicat drepturile asupra CP/M, Intel i-a acceptat cererea deoarece nu credea că microcalculatoarele cu disc rigid au un viitor. Ulterior, Kildall a format Digital Research pentru a dezvolta și vinde CP/M.

În 1977, Digital Research a reproiectat CP/M pentru a-l face potrivit pentru microcalculatoarele cu Intel 8080, Zilog Z80 și alte procesoare. Multe programe de aplicații au fost scrise pentru a rula pe CP/M, permițând sistemului să dețină poziția de top în lumea microcalculatoarelor timp de 5 ani.

La începutul anilor 1980, IBM a dezvoltat IBM PC (Personal Computer – calculator personal)[3] și a început să caute software pentru acesta. Personalul IBM l-a contactat pe Bill Gates pentru a obține o licență de utilizare a interpretorului său pentru limbajul BASIC. De asemenea, l-au întrebat dacă poate știe de un sistem de operare care să funcționeze pe un IBM PC. Gates i-a sfătuit să meargă la Digital Research, pe atunci principala companie de sisteme de operare. Dar Kildall a refuzat să se întâlnească cu IBM, trimițându-și în schimb subalternul. Pentru a înrăutăți lucrurile, avocatul său a refuzat chiar să semneze un acord de confidențialitate în ceea

[3] Este important de reținut că, în acest caz, "IBM PC" este numele unui anumit model de calculator, în timp ce "PC" poate fi considerat ca o abreviere pentru "Personal Computer", o denumire a unei clase de calculatoare. Denumirea consacrată a acestui model ca fiind pur și simplu "PC" nu este corectă - în aceeași perioadă existau și alte calculatoare personale.

ce privește IBM PC-ul care urma să fie lansat, distrugând complet cazul. Corporația IBM i-a cerut din nou lui Gates să îi furnizeze un sistem de operare. După o a doua solicitare, Gates a aflat că un producător local de calculatoare, Seattle Computer Products, avea un sistem DOS (Disk Operating System). S-a dus la acea companie cu o ofertă de a cumpăra DOS (ar fi fost vorba de $50.000), pe care Seattle Computer Products a acceptat-o cu bucurie. Gates a creat apoi pachetul software DOS/BASIC, care a fost cumpărat de IBM. Când IBM a vrut să aducă unele îmbunătățiri la sistemul de operare, Bill Gates l-a invitat pe Tim Paterson, cel care a scris DOS și care a devenit primul angajat al Microsoft, compania nou înființată de Gates. Sistemul modificat a fost redenumit în MS-DOS (MicroSoft Disk Operating System) și a devenit rapid dominant pe piața PC-urilor IBM. Decisivă a fost decizia lui Gates (extrem de înțeleaptă) să vândă sistemul de operare MS-DOS companiilor de calculatoare pentru a fi instalat împreună cu hardware-ul lor, spre deosebire de încercarea lui Kildall de a vinde CP/M direct utilizatorilor finali (cel puțin la început).

Atunci când în anul 1983 a apărut IBM PC/AT (o dezvoltare ulterioară a familiei IBM PC) cu un procesor Intel 80286, MS-DOS era deja bine stabilit, iar CP/M își ducea ultimele sale zile. Ulterior, MS-DOS a fost utilizat pe scară largă pe calculatoarele cu procesoarele 80386 și 80486. Deși versiunea originală a MS-DOS era mai degrabă primitivă, versiunile ulterioare erau mult mai avansate, multe dintre acestea fiind împrumutate de la UNIX. (Corporația Microsoft cunoștea foarte bine SO UNIX, în primii ani chiar a vândut o versiune UNIX pentru microcalculatoare - XENIX.)

CP/M, MS-DOS și alte sisteme de operare pentru primele microcalculatoare erau bazate în întregime pe comenzi de la tastatură. Cu timpul, datorită cercetărilor efectuate în anii 1960 de Doug Engelbart de la Stanford Research Institute, această situație s-a schimbat. Engelbart a inventat interfața grafică cu utilizatorul (GUI) cu ferestre, pictograme, sisteme de meniuri și mouse-ul. Cercetătorii de la Xerox PARC au preluat această idee și au folosit-o în mașinile pe care le-au construit.

Într-o zi, Steve Jobs, unul dintre autorii computerului Apple proiectat în garajul său, a vizitat PARC, unde a văzut GUI-ul și a realizat imediat potențialul pe care aceasta o deținea, potențial subestimat de conducerea Xerox. Jobs a demarat apoi crearea unui computer Apple echipat cu interfața grafică. Acest proiect a dus la crearea computerului Lisa, care a fost prea scump și nu a avut succes comercial. A doua încercare a lui Jobs, computerul Apple Macintosh, a fost un succes nu numai pentru că era mult mai ieftin decât Lisa, ci și pentru că avea o interfață de utilizator mai prietenoasă, concepută pentru utilizatorii care nu erau familiarizați cu computerele și care nu erau dornici să învețe nimic. Macintosh a cunoscut o susținere semnificativă în rândul persoanelor creative - designeri grafici, fotografi digitali profesioniști și companii profesionale de producție video digitală - care l-au adoptat. În 1999, Apple a împrumutat nucleul derivat din microkernelul Mach dezvoltat inițial de Universitatea Carnegie Mellon pentru a înlocui nucleul BSD UNIX. Prin urmare, Mac OS X este un sistem de operare bazat pe UNIX, deși cu o interfață foarte specială.

Când Microsoft a decis să creeze un succesor pentru MS-DOS, a fost foarte impresionată de succesul lui Macintosh. Rezultatul a fost un sistem bazat pe o interfață grafică cu utilizatorul, numită Windows, care a fost inițial un add-on pentru MS-DOS (adică mai mult un shell decât un sistem de operare propriu-zis). Timp de aproximativ 10 ani, din 1985 până în 1995, Windows a fost doar un shell grafic peste MS-DOS. Dar, în 1995 a fost lansată o versiune independentă Windows - SO Windows 95. Acesta îndeplinea direct majoritatea funcțiilor sistemului de operare, folosind sistemul MS-DOS inclus doar pentru a porni și a rula programe vechi concepute pentru MS-DOS. În 1998, a fost lansată o versiune ușor modificată a sistemului,

numită Windows 98. Cu toate acestea, atât Windows 95, cât și Windows 98 conțineau încă o cantitate destul de mare de cod de asamblare scris pentru procesoarele Intel pe 16 biți.

Celălalt sistem de operare Microsoft a fost Windows NT (NT înseamnă New Technology), care era, într-o anumită măsură, compatibil cu Windows 95. Windows NT a fost rescris și a devenit un sistem complet pe 32 de biți. Principalul dezvoltator al Windows NT a fost David Cutler, co-dezvoltătorul sistemului de operare VAX VMS, astfel încât unele idei din VMS sunt prezente în NT (atât de mult încât proprietarul VMS, DEC, a dat în judecată Microsoft. Conflictul a fost soluționat pe cale amiabilă pentru o sumă decentă). Microsoft se aștepta ca prima versiune să înlocuiască MS-DOS și toate celelalte versiuni de Windows, deoarece era mult superioară, dar așteptările nu au fost îndeplinite. Doar Windows NT 4.0 a reușit în cele din urmă să câștige o popularitate ridicată, în special în rețelele corporative. Cea de-a cincea versiune a Windows NT a fost redenumită Windows 2000 la începutul anului 1999. Aceasta a fost destinată să înlocuiască atât Windows 98, cât și Windows NT 4.0.

Dar aceste planuri nu erau sortite să se împlinească în totalitate, așa că Microsoft a lansat o altă versiune de Windows 98, numită Windows Me (Millennium edition). În 2001 a fost lansată o versiune ușor actualizată a Windows 2000, numită Windows XP. Această versiune a fost lansată pentru o perioadă mult mai lungă, înlocuind practic toate versiunile anterioare de Windows.

Cu toate acestea, noi versiuni au continuat să fie lansate. După Windows 2000, Microsoft a împărțit familia Windows într-o linie de produse pentru clienți și una pentru servere. Linia bazată pe client se baza pe XP și succesorii săi, în timp ce linia bazată pe server era formată din Windows Server 2003 și Windows 2008. Puțin mai târziu a apărut o a treia linie destinată lumii sistemelor de operare incorporate. Variațiile sub formă de service pack-uri au fost separate de toate aceste versiuni de Windows. Acest lucru a fost suficient pentru a-i liniști pe unii administratori (și pe autorii manualelor de sisteme de operare).

Apoi, în ianuarie 2007, Microsoft a lansat o versiune finală a succesorului Windows XP, numită Vista. Avea o nouă interfață grafică, o securitate îmbunătățită și multe programe de utilizator noi sau actualizate. Microsoft a sperat că va înlocui complet Windows XP, dar nu a reușit. În schimb, a primit numeroase critici și articole de presă, în special din cauza cerințelor de sistem ridicate, a condițiilor restrictive de acordare a licențelor și a suportului pentru tehnologia de protecție a drepturilor de autor (tehnologie care îngreunează copierea de către utilizatori a materialelor protejate).

Odată cu apariția lui Windows 7, un sistem de operare nou și mai puțin pretențios, mulți au decis să renunțe complet la Vista. Windows 7 nu a introdus multe caracteristici noi, dar a fost relativ mic și destul de stabil. Windows 7 a câștigat mai multă cotă de piață în mai puțin de trei săptămâni decât a câștigat Vista în șapte luni. În 2012, Microsoft a lansat succesorul lui Windows 7, Windows 8 - un sistem de operare cu un aspect complet nou, conceput pentru ecrane tactile. Compania a sperat că noul design va face din sistemul de operare o alegere dominantă pentru o gamă largă de dispozitive: desktop-uri, laptop-uri, tablete, telefoane și computere personale folosite ca home theater. Dar, până în prezent, pătrunderea sa pe piață a fost mult mai lentă decât în cazul Windows 7.

Celălalt concurent principal din lumea PC-urilor este sistemul de operare UNIX (și diferitele sale derivate). UNIX are o poziție mai puternică pe serverele de rețea și industriale, dar devine din ce în ce mai răspândit și pe desktop-uri, laptop-uri, tablete și smartphone-uri. Pe computerele cu procesor Pentium, sistemul de operare Linux devine o alternativă populară la Windows pentru studenți și pentru un număr tot mai mare de utilizatori corporativi.

În această carte, termenul x86 va fi aplicat tuturor procesoarelor moderne bazate pe familia de arhitecturi cu seturi de instrucțiuni provenite de la procesorul 8086 creat în anii 1970. Companiile AMD și Intel au produs o mulțime de astfel de procesoare care, de multe ori, se deosebesc considerabil: procesoarele pot fi pe 32 sau 64 de biți, cu un număr mic sau mare de nuclee, cu o adâncime diferită a conveierilor etc. Totuși, ele sunt destul de asemănătoare pentru un programator și toate pot rula codul unui procesor 8086 scris acum 35 de ani. În cazul în care diferențele joacă un rol important, se vor face referiri la modele specifice, iar termenii x86-32 și x86-64 vor fi utilizați pentru a indica variantele pe 32 și 64 de biți.

Sistemul de operare FreeBSD este, de asemenea, un derivat popular al SO UNIX, creat în cadrul proiectului BSD de la Berkeley. Toate computerele Macintosh moderne rulează o versiune modificată a FreeBSD (OS X). UNIX este, de asemenea, SO standard pe stațiile de lucru echipate cu procesoare RISC de înaltă performanță. Derivatele sale au fost utilizate pe scară largă pe dispozitivele mobile care rulează iOS 7 sau Android.

În timp ce mulți utilizatori UNIX, în special programatori experimentați, preferă o interfață bazată pe linia de comandă, aproape toate sistemele UNIX acceptă sistemul X Window (sau X11), creat la Massachusetts Institute of Technology. Acest sistem efectuează operațiuni de bază de control al ferestrelor, permițând utilizatorilor să creeze, să șteargă, să mute și să redimensioneze ferestrele cu ajutorul mouse-ului. Adesea, o interfață grafică completă pentru utilizator, cum ar fi Gnome sau KDE, poate fi utilizată ca un add-on la X11, conferind UNIX un aspect și o senzație oarecum asemănătoare cu Macintosh sau Microsoft Windows.

La mijlocul anilor 1980, a început să se dezvolte un fenomen interesant - creșterea rețelelor de calculatoare personale care rulează sisteme de operare de rețea și sisteme de operare distribuite (Tanenbaum și Van Steen, 2007). În sistemele de operare de rețea, utilizatorii sunt conștienți de existența mai multor calculatoare și se pot conecta la un calculator la distanță și pot copia fișiere de pe un calculator pe altul. Fiecare mașină rulează propriul sistem de operare local și are propriul utilizator(i) local(i).

Sistemele de operare de rețea nu sunt semnificativ diferite de sistemele de operare cu un singur procesor. În mod evident, acestea au nevoie de un controler de interfață de rețea și de un software de nivel scăzut pentru a opera acest controler, precum și de programe pentru a se conecta la mașina de la distanță și a accesa fișiere de la distanță, dar aceste adăugiri nu schimbă structura de bază a sistemului de operare.

În schimb, un sistem de operare distribuit se prezintă utilizatorilor săi ca un sistem tradițional cu un singur procesor, când, de fapt, este format din mai multe procesoare. Utilizatorii nu trebuie să știe unde rulează programele lor sau unde se află fișierele lor - toate acestea trebuie să fie gestionate automat și eficient de către sistemul de operare însuși.

Adevăratele sisteme de operare distribuite necesită mult mai multe modificări decât simpla adăugare a unei cantități mici de cod la un sistem de operare cu un singur procesor, deoarece sistemele distribuite și cele centralizate sunt foarte diferite. De exemplu, sistemele distribuite permit adesea ca aplicațiile să ruleze simultan pe mai multe procesoare, ceea ce necesită algoritmi mai complecși pentru a distribui munca procesoarelor în vederea optimizării gradului de procesare paralelă a datelor. Prezența latenței (întârzierilor) în transmiterea datelor în rețea implică adesea faptul că acești (și alți) algoritmi trebuie să funcționeze în condiții de informații incomplete, depășite sau chiar incorecte. Această situație este fundamental diferită de cea a unui sistem cu un singur procesor.

1.2.5 Generația a cincea (din 1990 până în prezent): calculatoarele mobile

Încă de pe timpurile benzilor desenate din anii 1940 când detectivul Dick Tracy a început să comunice folosind un radioemițător montat în ceasul de mână, oamenii au dorit un dispozitiv de comunicare pe care să-l poată purta cu ei oriunde mergeau. Primul telefon mobil adevărat a apărut în 1946 și cântărea aproximativ 40 de kilograme. Îl puteai lua cu tine oriunde te duceai, dacă aveai o mașină în care să îl transporți.

Primul telefon portabil cu adevărat a apărut în anii 1970 și, cântărind aproximativ un kilogram, a fost acceptat relativ pozitiv. Era cunoscut cu afecțiune sub numele de "cărămidă". Curând, toată lumea și-a dorit unul. Astăzi, gradul de penetrare a telefoanelor mobile este de aproape 90% din populația globală. Putem efectua apeluri nu doar cu telefoanele noastre portabile și cu ceasurile de mână, ci în curând și cu ochelarii și alte obiecte portabile. În plus, partea cu telefonul nu mai este atât de interesantă. Primim e-mailuri, navigăm pe internet, trimitem mesaje text prietenilor noștri, ne jucăm în jocuri, și aflăm despre cât de aglomerat este traficul fără să ne mai mire toate astea.

Deși ideea de a combina telefonul și calculatorul într-un singur dispozitiv exista încă din anii 1970, primul smartphone real a apărut abia la mijlocul anilor 1990, când Nokia a lansat N9000, care a combinat la propriu două dispozitive în mare parte separate: un telefon și un PDA (Personal Digital Assistant). În 1997, Ericsson a inventat termenul de smartphone pentru modelul său GS88 "Penelope".

Acum, când smartphone-urile au devenit omniprezente, concurența dintre diferite sisteme de operare este acerbă, iar rezultatul este chiar mai puțin clar decât în lumea PC-urilor. Când sunt redactate aceste rânduri, Android de la Google este sistemul de operare dominant, iar iOS de la Apple este clar pe locul al doilea, dar nu a fost întotdeauna așa și s-ar putea ca totul să fie din nou diferit în doar câțiva ani. Dacă ceva este clar în lumea smartphone-urilor, este faptul că nu este ușor să rămâi regele muntelui pentru mult timp.

Majoritatea smartphone-urilor din primul deceniu de la înființare rulau Symbian OS. Acesta a fost sistemul de operare ales de mărci populare precum Samsung, Sony Ericsson, Motorola și, mai ales, Nokia. Cu toate acestea, alte sisteme de operare precum Blackberry OS de la RIM (introdus pentru smartphone-uri în 2002) și iOS de la Apple (lansat pentru primul iPhone în 2007) au început să preia cota de piață de la Symbian. Mulți se așteptau ca RIM să domine piața de afaceri, în timp ce iOS va fi regele dispozitivelor de consum. Cota de piață a Symbian s-a prăbușit. În 2011, Nokia a renunțat la Symbian și a anunțat că se va concentra pe Windows Phone ca platformă principală. Pentru o perioadă, Apple și RIM au fost cele mai tari (deși nu la fel de dominante precum fusese Symbian), dar nu a durat foarte mult până când Android, un sistem de operare bazat pe Linux lansat de Google în 2008, să își depășească toți rivalii.

Pentru producătorii de telefoane, Android avea avantajul de a fi open source și disponibil sub o licență permisivă. Prin urmare, aceștia puteau să îl modifice și să-l adapteze cu ușurință la propriul hardware. De asemenea, dispune de o comunitate uriașă de dezvoltatori care scriu aplicații, majoritatea în limbajul de programare Java, care le este familiar. Chiar și așa, ultimii ani au arătat că această dominație ar putea să nu dureze, iar concurenții Android sunt dornici să recupereze o parte din cota de piață. Vom analiza Android în detaliu în secțiunea 10.8.

1.3 Notiuni din domeniul resurselor fizice ale calculatorului

Un sistem de operare este strâns legat de hardware-ul computerului pe care rulează. Acesta extinde setul de instrucțiuni al computerului și îi gestionează resursele. Pentru corecta funcționare sunt necesare cunoștințe adânci despre hardware, cel puțin despre modul în care

acesta apare în ochii programatorului. Din acest motiv, haideți să trecem în revistă pe scurt resurselor fizice ale calculatorului care intră în componența calculatoarelor personale moderne. După aceasta vom putea începe studiul detaliat a ceea ce fac și cum funcționează sistemele de operare.

Din punct de vedere conceptual, un computer personal simplu poate fi abstractizat la un model asemănător cu cel din figura 1-6.

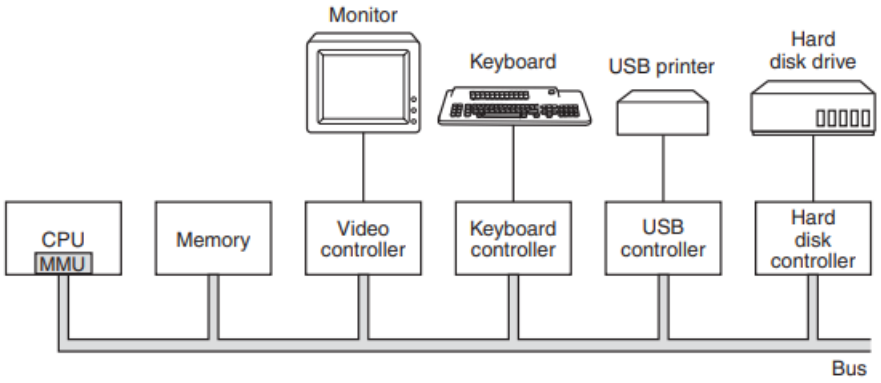

Fig. 1.6. Componentele unui calculator personal

Unitatea centrală, memoria și dispozitivele I/O sunt conectate cu ajutorul magistralei (bus) de sistem și comunică între ele prin intermediul acesteia. Calculatoarele personale moderne au o structură mai complicată, care implică mai multe magistrale, pe care le vom examina mai târziu. Deocamdată, acest model va fi suficient. În secțiunile următoare, vom trece în revistă pe scurt aceste componente și vom examina unele dintre problemele hardware care îi preocupă pe proiectanții de sisteme de operare. Inutil să mai spunem că acesta va fi un rezumat foarte compact. S-au scris multe cărți pe tema hardware-ului și a organizării calculatoarelor. Două dintre cele mai cunoscute sunt Tanenbaum și Austin (2012) și Patterson și Hennessy (2013).

1.3.1 Procesoarele

Unitatea centrală de procesare (procesorul) este 'creierul" calculatorului. Acesta preia instrucțiunile din memorie și le execută. Ciclul de bază al fiecărei unități centrale de procesare constă din (1) preluarea unei instrucțiuni din memorie, (2) decodificarea acesteea pentru a-i determina tipul și operanzii și (3) executarea. Ciclul se repetă pornind de la prima instrucțiune (punctul de intrare în program) până când programul se termină. În acest fel sunt executate programele.

Fiecare unitate centrală de procesare are un set specific de instrucțiuni pe care le poate executa. Astfel, un x86 nu poate executa programe ARM, iar un procesor ARM nu poate executa programe x86. Deoarece accesarea memoriei pentru a obține o instrucțiune sau un cuvânt de date necesită mult mai mult timp decât executarea unei instrucțiuni, toate CPU-urile conțin unele registre în interior pentru a păstra variabilele cheie și rezultatele temporare. Astfel, setul de instrucțiuni conține, în general, instrucțiuni pentru a încărca un cuvânt din memorie într-un registru și pentru a stoca un cuvânt dintr-un registru în memorie. Alte instrucțiuni combină doi operanzi din registre, din memorie sau din ambele într-un rezultat, cum ar fi adunarea a două cuvinte și stocarea rezultatului într-un registru sau în memorie.

În afară de registrele generale utilizate pentru a păstra variabilele și rezultatele temporare, majoritatea calculatoarelor dispun de mai multe registre speciale care sunt accesibile programatorului. Unul dintre acestea este contorul de instrucțiuni, care conține adresa de memorie a următoarei instrucțiuni care urmează să fie preluată. După ce instrucțiunea

respectivă a fost preluată, contorul de program este actualizat pentru a indica adresa următoarei instrucțiuni.

Un alt registru, numit pointer de stivă, indică topul stivei curente din memorie. Stiva conține câte un cadru (zonă de memorie) pentru fiecare procedură în care programul a intratat, dar din care nu a ieșit încă. Cadrul de stivă (mediul procedurii curente) al unei proceduri conține parametrii de intrare, variabilele locale și variabilele temporare care nu sunt păstrate în registre.

Un alt registru este PSW (Program Status Word). Acest registru conține biții de cod de stare, care sunt setați de instrucțiunile de comparare, biții pentru gestiunea priorității, modul (utilizator sau kernel) și diverși alți biți de control. Programele de utilizator pot citi, în mod normal, întregul PSW, dar, de obicei, pot scrie doar în unele dintre câmpurile acestuia. PSW joacă un rol important în apelurile de sistem și în operațiile de I/O.

Sistemul de operare trebuie să cunoască totul despre toate registrele. La multiplexarea temporală a unității centrale, sistemul de operare va opri adesea programul în curs de execuție pentru a (re)porni un altul. De fiecare dată când oprește un program în execuție, sistemul de operare trebuie să salveze toate registrele, astfel încât să poată fi restaurate atunci când va fi reluată execuția programului.

Pentru a îmbunătăți performanțele, proiectanții de unități centrale de procesare au abandonat de mult timp modelul simplu de preluare, decodificare și executare a câte unei instrucțiuni pe rând. Multe unități centrale de procesare moderne dispun de facilități pentru a executa mai mult de o instrucțiune în același timp. De exemplu, o unitate centrală poate avea unități separate de preluare, decodificare și execuție, astfel încât, în timp ce execută instrucțiunea *n*, ar putea, de asemenea, să decodifice instrucțiunea *n* + 1 și să preia instrucțiunea *n* + 2. O astfel de organizare se numește pipeline și este ilustrată în Fig. 1-7(a) pentru un pipeline cu trei etape. De obicei pipeline-urile sunt mai lungi. În cele mai multe modele de pipeline, odată ce o instrucțiune a fost preluată în pipeline, aceasta trebuie executată, chiar dacă instrucțiunea precedentă a fost o ramificare condiționată. Pipeline-urile dau mari bătăi de cap dezvoltatorilor de compilatoare și de sisteme de operare, deoarece aceștia sunt expuși complexității mașinii fizice și sunt obligați să rezolve problemele care apar aici.

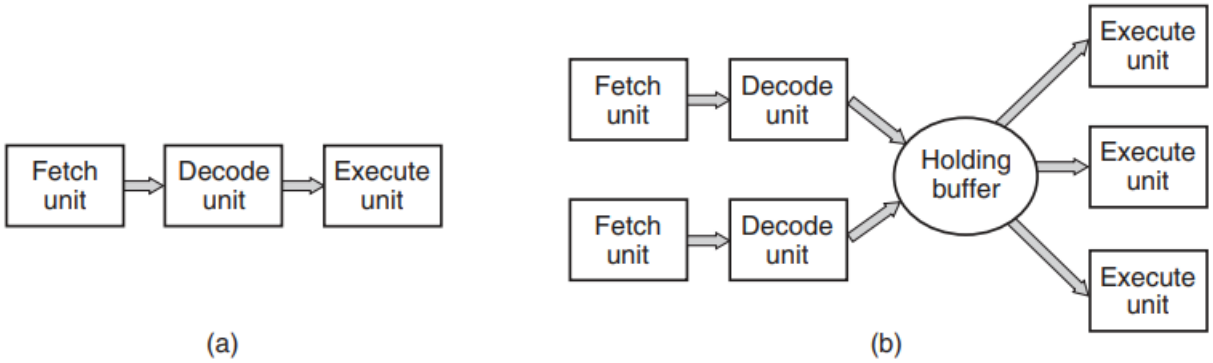

Fig. 1.7. (a) Pipeline cu trei etape. (b) Unitate centrală superscalară

Procesorul superscalar, prezentat în figura 1.7, b, are un design mai avansat decât procesorul pipeline. Acesta are mai multe unități de execuție, de exemplu: una pentru aritmetica numerelor întregi, una pentru aritmetica în virgulă mobilă și una pentru operațiile logice. Două sau mai multe instrucțiuni sunt preluate în același timp, decodificate și plasate într-un buffer de stocare în așteptarea executării. De îndată ce unitatea de execuție devine disponibilă, aceasta caută în memoria tampon de stocare o instrucțiune pe care o poate executa și, dacă este disponibilă, o preia din memoria tampon și o execută. Ca urmare, comenzile programului adesea nu respect

ordinea de execuție. De asigurarea corectitudinii rezultatului final (că acesta va fi același cu cel care rezultă din respectarea ordinei de execuție) sunt reposnabile, de obicei, resursele tehnice. Însă, după cum vom vedea mai târziu, această abordare introduce complicații neplăcute în sistemul de operare.

După cum s-a menționat, majoritatea procesoarelor, cu excepția celor mai simple utilizate în sistemele incorporate, au două moduri de funcționare: modul kernel și modul utilizator. De obicei, modul este controlat de un bit special din cuvântul de stare a programului, PSW. Atunci când funcționează în modul kernel, procesorul poate executa orice comandă din setul său de comenzi și poate utiliza orice posibilități ale resurselor fizice. Pe desktopuri și pe calculatoarele de tip server, sistemul de operare rulează de obicei în modul kernel, ceea ce îi oferă acces la toate capacitățile resurselor fizice. Pe majoritatea sistemelor incorporate, doar o mică parte a sistemului de operare rulează în modul kernel, restul sistemului de operare rulând în modul utilizator.

Programele de utilizator rulează întotdeauna în modul utilizator, adică sunt executate doar instrucțiuni din cadrul unei submulțini de comenzi (numite nepriveligiate) și au acces la un subset de capacități ale hardware-ului. De regulă, toate comenzile referitoare la operațiunile de I/O și la protecția memoriei sunt interzise în modul utilizator. De asemenea, este bineînțeles interzisă setarea modului kernel prin modificarea valorii bitului PSW.

Pentru a obține servicii de la sistemul de operare, programul utilizatorului trebuie să genereze un **apel de sistem** care este interceptat în kernel și care apelează sistemul de operare. Instrucțiunea de interceptare *TRAP hook* realizează comutarea din modul utilizator în modul kernel și pornește sistemul de operare. Când procesarea apelului este finalizată, controlul este returnat programului utilizator și se execută comanda care urmează apelului de sistem. Detaliile mecanismului de apelare a sistemului vor fi tratate mai târziu în acest capitol, dar pentru moment ar trebui să fie considerat un tip special de instrucțiune de apelare a procedurii care are proprietatea suplimentară de a trece din modul utilizator în modul kernel. De acum încolo, apelurile de sistem vor fi evidențiate cu același font ca în acest cuvânt: ***read***.

Desigur, calculatoarele au și alte întreruperi de sistem care nu sunt concepute pentru a intercepta instrucțiunea de apel de sistem. Dar majoritatea celorlalte întreruperi de sistem sunt declanșate de întreruperi pentru a avertiza asupra unor situații excepționale, cum ar fi încercările de împărțire la zero sau dispariția unui ordin în timpul unei operații în virgulă mobilă. În toate cazurile, controlul se transferă către sistemul de operare, care trebuie să decidă ce trebuie să facă în continuare. Uneori, programul trebuie să fie încheiat printr-un mesaj de eroare. În alte cazuri, eroarea poate fi ignorată (de exemplu, atunci când ordinul unui număr dispare, se poate presupune că acesta este zero). În sfârșit, atunci când programul a declarat în prealabil că va gestiona singur unele dintre situațiile posibile, controlul ar trebui să fie returnat programului pentru ca acesta să poată rezolva singur problema.

1.3.2 Cipuri multi-fire și multi-nuclee

Legea lui Moore afirmă că numărul de tranzistori într-un cip se dublează la fiecare 18 luni. Această "lege" nu este un fel de lege a fizicii, cum ar fi conservarea impulsului, ci este o observație a cofondatorului Intel, Gordon Moore, cu privire la rapiditatea cu care inginerii de la companiile de semiconductori reușesc să micșoreze tranzistorii. Legea lui Moore este valabilă de peste trei decenii și se preconizează că va mai fi valabilă cel puțin încă unul. După aceea, numărul de atomi pe tranzistor va deveni prea mic, iar mecanica cuantică va începe să joace un rol important, împiedicând reducerea în continuare a dimensiunilor tranzistorului.

Abundența tranzistoarelor duce la o problemă: ce să facem cu toate acestea? Am văzut mai sus o abordare: arhitecturi superscalare, cu unități funcționale multiple. Dar, pe măsură ce numărul de tranzistori într-o unitate de volum crește, sunt posibile și mai multe unități funcționale. Un lucru evident de făcut este să punem memorii cache mai mari pe cipul procesorului. Aceasta se și întâmplă, dar, în cele din urmă, se va ajunge la punctul de descreștere a randamentului.

Următorul pas evident este replicarea nu numai a unităților funcționale, ci și a unei părți din logica de control. Intel Pentium 4 a introdus această proprietate, numită multithreading sau hyperthreading (denumirea Intel), în procesorul x86, ca și în cazul altor câteva cipuri de procesor, inclusiv SPARC, Power5, Intel Xeon și familia Intel Core. Într-o primă aproximație, această tehnologie permite procesorului să păstreze starea a două fire diferite și să se comuteze între ele timp de câteva nanosecunde. (Un fir este un fel de proces ușor, care, la rândul său, este un program în curs de execuție; vom intra în detalii în cap. 2). De exemplu, dacă unul dintre procese trebuie să citească un cuvânt din memorie (ceea ce durează multe cicluri de ceas), un CPU cu mai multe fire poate pur și simplu să treacă la un alt fir. Multithreading-ul nu oferă un paralelism adevărat. Doar un singur proces la un moment dat rulează, dar timpul de comutare a firelor este redus la ordinul unor nanosecunde.

Multithreading-ul are implicații pentru sistemul de operare, deoarece fiecare fir apare pentru sistemul de operare ca o unitate centrală de procesare separată. Să ne închipuim un sistem cu două procesoare reale, fiecare cu două fire de execuție. Sistemul de operare va vedea patru procesoare. Dacă există lucru doar pentru a menține ocupate două procesoare la un anumit moment de timp, acesta poate programa din greșeală două fire pe același processor fizic, iar celălalt processor să rămână complet inactiv. Această alegere este cu mult mai puțin eficientă decât utilizarea unui singur fir pe fiecare CPU.

Dincolo de multithreading, multe cipuri CPU au acum patru, opt sau mai multe procesoare sau nuclee complete pe ele. Cipurile multicore din Fig. 1-8 conțin efectiv patru miniprocesoare, fiecare cu propriul CPU independent. (Despre cache vom explica mai jos.) Unele procesoare, cum ar fi Intel Xeon Phi și Tilera TilePro, au deja mai mult de 60 de nuclee pe un singur cristal. Folosirea unui astfel de procesor multicore va necesita cu siguranță un sistem de operare multiprocesor.

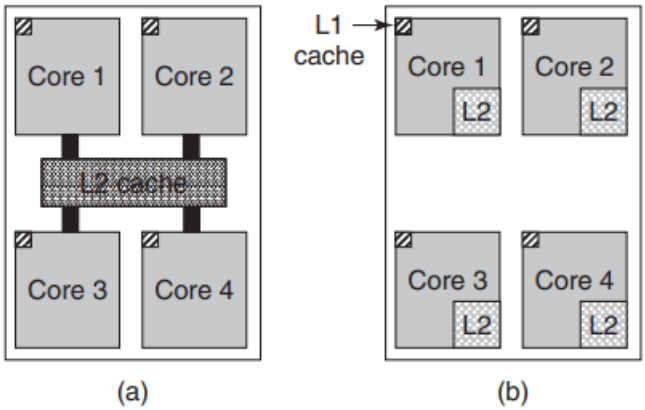

Fig. 1.8. (a) Un cip cu patru nuclee cu o memorie cache L2 partajată. (b) Un cip patru nuclee cu memorii cache L2 separate

De altfel, în ceea ce privește numărul absolut de nuclee, nimic nu întrece un GPU (unitate de procesare grafică) modern. Un GPU este un procesor cu, literalmente, mii de nuclee minuscule. Acestea sunt foarte bune pentru multe calcule mici efectuate în paralel, cum ar fi redarea

poligoanelor în aplicațiile grafice. Nu sunt la fel de bune pentru sarcini seriale. De asemenea, sunt greu de programat. În timp ce GPU-urile pot fi utile pentru sistemele de operare (de exemplu, criptarea sau procesarea traficului de rețea), este puțin probabil ca partea principală a sistemului de operare să ruleze pe astfel de procesoare.

1.3.3 Memoria

A doua componentă principală a oricărui calculator este memoria. În mod ideal, memoria ar trebui să fie cât mai rapidă (mai rapidă decât execuția unei singure instrucțiuni, astfel încât procesorul să nu fie încetinit de accesarea memoriei), destul de mare și extrem de ieftină. Nici o tehnologie modernă nu poate îndeplini toate aceste cerințe, astfel încât se folosește o altă abordare. Sistemul de memorie este conceput ca o ierarhie de niveluri (figura 1.9).

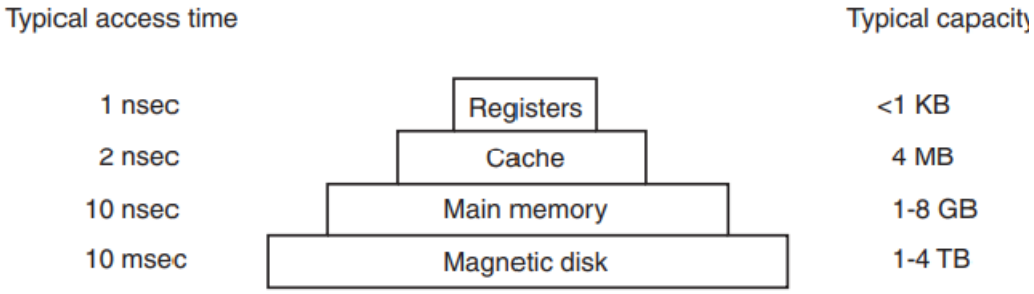

Fig. 1.9. Ierarhia unei memorii tipice. Valorile numerice sunt foarte aproximative.

Nivelurile superioare au o viteză mai mare, o capacitate mai mică și un cost unitar de stocare a unui bit de informație mai mare decât nivelurile inferioare, uneori de miliarde de ori.

Nivelul superior este format din registrele interne ale procesorului. Acestea sunt alcătuite din același material ca și procesorul și, prin urmare, sunt la fel de rapide. În consecință, nu există nicio întârziere în accesarea lor. Capacitatea de stocare disponibilă în ele este, de obicei, de 32 ×32 biți la un CPU pe 32 de biți și de 64×64 biți la un CPU pe 64 de biți. În ambele cazuri acest volum nu depășește vloarea de 1 KB. Programele singure pot să gestioneze registrele (adică să decidă ce să păstreze în ele) fără implicarea hardware-ului.

Urmează memoria cache, care este în mare parte controlată de hardware. Memoria principală este împărțită în linii de memorie cache, de obicei câte 64 octeți, cu adresele 0 până la 63 în linia cache 0, 64 până la 127 în linia cache 1, și așa mai departe. Cele mai utilizate linii de cache sunt păstrate într-o memorie cache de mare viteză situată în interiorul sau foarte aproape de CPU. Atunci când programul trebuie să citească un cuvânt de memorie, hardware-ul cache-ului verifică dacă nu cumva linia necesară se află în memoria cache. Dacă da, ceea ce se numește "cache hit", cererea este satisfăcută din memoria cache și nu este trimisă nicio cerere de memorie pe magistrala către memoria principală. În mod normal, accesarea cache-ului durează aproximativ două cicluri de ceas. În cazul lipsei cuvântului necesar în linia cache, trebuie să se meargă la memorie operativă, ceea ce va însemna întârziere suplimentară substanțială în timp. Memoria cache este limitată ca dimensiune din cauza costul ridicat. Unele mașini au două sau chiar trei niveluri de memorie cache, fiecare mai lent și mai mare decât cel anterior.

Memorizarea în cache joacă un rol esențial în multe domenii ale informaticii, nu doar pentru păstrarea șirurilor RAM. Caching-ul este utilizat frecvent pentru a îmbunătăți performanța atunci când există o resursă mare care poate fi împărțită în părți, dintre care unele sunt mult mai des utilizate decât altele. Sistemele de operare folosesc caching peste tot. De exemplu, majoritatea sistemelor de operare păstrează fișierele (sau părți de fișiere) foarte utilizate în memoria RAM,

evitând astfel să fie nevoie să le citească din nou în mod repetat de pe disc. Analogic, rezultatele conversiilor numelor lungi de fișiere, cum ar fi

`/home/ast/projects/minix3/src/kernel/clock.c,`

la adresa de disc unde se află fișierul pot fi stocate în memoria cache pentru a evita necesitatea unor căutări repetate. În cele din urmă, se poate stoca în memoria cache pentru utilizare ulterioară rezultatul conversiei dintre adresa unei pagini web (URL) și adresa de rețea (adresa IP). Există multe alte utilizări ale tehnologiei de stocare în memoria cache.

În orice sistem de caching apar destul de repede o serie de probleme.

1. Când se plasează un element nou în memoria cache?
2. În ce zonă de memorie cache ar trebui plasat un nou element?
3. Ce intrare să fie eliminată (“victima”) din memoria cache atunci când dorim să obținem spațiu liber?
1. Unde anume va fi plasată “victima” (în memoria secundară)?

Nu toate aceste întrebări au legătură cu memoria cache. De exemplu, în procesul de stocare șirurilor de caractere RAM în memoria cache a procesorului, un nou element va fi de obicei introdus în memoria cache de fiecare dată când accesarea memoriei cache a fost fără succes. Atunci când se calculează linia de memorie cache dorită pentru a plasa un nou element, de obicei sunt utilizați unii dintre biții superiori ai acelei adrese de memorie, la care se face referință. De exemplu, dacă există 4096 de linii de memorie cache de 64 de octeți și adrese de 32 de biți, biții de la 6 la 17 ar putea fi utilizați pentru a determina linia de memorie cache, iar biții de la 0 la 5 ar putea fi utilizați pentru a determina octetul din linia de memorie cache. În acest caz, elementul care urmează să fie șters este același cu cel care conține noile date, dar în alte sisteme este posibil ca această ordine să nu fie respectată. În sfârșit, atunci când o linie de memorie cache este suprascrisă în memoria RAM (dacă a fost modificată în timpul procesului de caching), locația de memorie în care urmează să fie suprascrisă este definită explicit prin adresa din cerere.

Utilizarea memoriei cache s-a dovedit a fi atât de reușită încât multe procesoare moderne au două niveluri de memorie cache. Primul nivel, sau memoria cache L1, face întotdeauna parte din procesor și, de obicei, furnizează instrucțiuni decodate motorului de execuție a instrucțiunilor procesorului. Multe procesoare au o a doua memorie cache L1 pentru acele cuvinte de date care sunt foarte utilizate.

În mod obișnuit, fiecare dintre memoriile cache L1 are o dimensiune de 16 Kbyte. În plus față de această memorie cache, procesoarele au adesea un al doilea nivel de memorie cache numit L2 cache, care conține câțiva megabytes de cuvinte de memorie utilizate recent. Diferența dintre memoria cache L1 și L2 este reprezentată de diagrama de sincronizare. Accesul la memoria cache de prim nivel nu are întârziere, în timp ce accesul la memoria cache de al doilea nivel necesită o întârziere de unul sau două cicluri de ceas.

Atunci când proiectează procesoare multi-core, proiectanții trebuie să decidă unde să plaseze memoria cache. Figura 1.8 (a) prezintă o singură memorie cache L2 partajată de toate nucleele. Această abordare este utilizată în procesoarele multi-core de la Intel. Pentru comparație, figura 1.8 (b) arată că fiecare nucleu are propriul cache L2. Aceasta este abordarea utilizată de AMD. Fiecare abordare are propriile sale argumente pro și contra. De exemplu, memoria cache L2 partajată de la Intel necesită controler de cache mai complex, în timp ce calea aleasă de AMD face dificilă menținerea unei stări consecvente a memoriei cache L2 între nuclee.

Următorul în ierarhia prezentată în figura 1.9 este memoria operativă. Aceasta este principala zonă de lucru a sistemului de memorie al mașinii. Memoria operativă este adesea denumită memorie cu acces aleatoriu (Random Access Memory (RAM)). Veteranii o numesc uneori memorie cu miez, deoarece în anii 1950 și 1960, în memoria RAM se foloseau mici miezuri de ferită magnetizabile. Au trecut decenii, dar numele s-a păstrat. În zilele noastre, blocurile de memorie au volume diferite de la sute de megaocteți la mai mulți gigaocteți, iar acest volum crește rapid. Toate cererile procesorului care nu pot fi satisfăcute de memoria cache sunt direcționate către la RAM.

Pe lângă memoria RAM, multe calculatoare sunt echipate cu o mică memorie cu acces aleatoriu, nevolatilă, numită *Memorie numai pentru citire* (ROM). Spre deosebire de RAM, ROM nu-și pierde conținutul atunci când se întrerupe alimentarea cu energie și, prin urmare, este nevolatilă. ROM-ul este programat din fabrică și nu poate fi modificat ulterior. Acest tip de memorie se caracterizează prin viteză mare și costuri reduse. Unele computere au un încărcător de pornire inițial în memoria ROM care este utilizat pentru a porni calculatorul. Unele controlere I/O au, de asemenea, acest tip de memorie pentru controlul de nivel scăzut al dispozitivului.

Există și alte tipuri de memorie nevolatilă care pot fi șterse și rescrise, spre deosebire de ROM-uri: memoriile permanente programabile șterse electric (EEPROM), cunoscute și sub numele de (Electrically Erasable PROM) și memoria flash. Deoarece scrierea EEPROM-urilor durează cu câteva ordine de mărime mai mult decât a RAM-urilor, acestea sunt utilizate în același scop ca și ROM. De asemenea, acestea au și caracteristica suplimentară de a putea corecta erorile din programele pe care le conțin prin suprascrierea spațiului de memorie pe care îl ocupă.

Memoria flash este utilizată în mod obișnuit ca suport de stocare pentru dispozitivele electronice portabile. Menționez doar două domenii de utlizare: servește drept "film (peliculă)" în aparatele foto digitale și "disc" în playerele muzicale portabile. în ceea ce privește performanța memoria flash se află la mijloc înttre RAM și disc. De asemenea, spre deosebire de un dispozitiv de stocare de tip "disc", dacă memoria flash este ștearsă sau suprascrisă de prea multe ori, aceasta devine vulnerabilă.

Un alt tip de memorie este memoria CMOS, care este nevolatilă. Multe calculatoare folosesc memoria CMOS pentru a stoca data și ora curentă. Memoria CMOS și circuitele electronice de cronometrare a timpului sunt alimentate de o baterie miniaturală (sau de un pachet de baterii), astfel încât ora curentă este întotdeauna actualizată chiar și atunci când calculatorul este deconectat de la o sursă de alimentare externă. Memoria CMOS poate stoca, de asemenea, parametrii de configurare care să indice, de exemplu, de la ce unitate trebuie să pornească sistemul. Consumul de energie al memoriei CMOS este atât de redus încât utilizarea unei baterii instalată din fabrică durează adesea mai mulți ani. Însă, atunci când bateria începe să cedeze, computerul poate da semne de "Alzheimer" și poate "uita" lucruri pe care le ținea minte de ani de zile, cum ar fi de exemplu de pe care hard disk să pornească.

1.3.4 Discurile

Următorul nivel al ierarhiei memoriei, după RAM, este discul magnetic (hard disk). O unitate de disc este de douzeci de ori mai ieftin decât memoria RAM în termeni de biți de informație, iar capacitatea de stocare este cu mult mai mare. Singura problema este că accesul la date este cu aproximativ trei ordine de mărime mai lent. Acest lucru se datorează faptului că este un dispozitiv mecanic a cărui construcție este prezentată în mod convențional în figura 1.10.

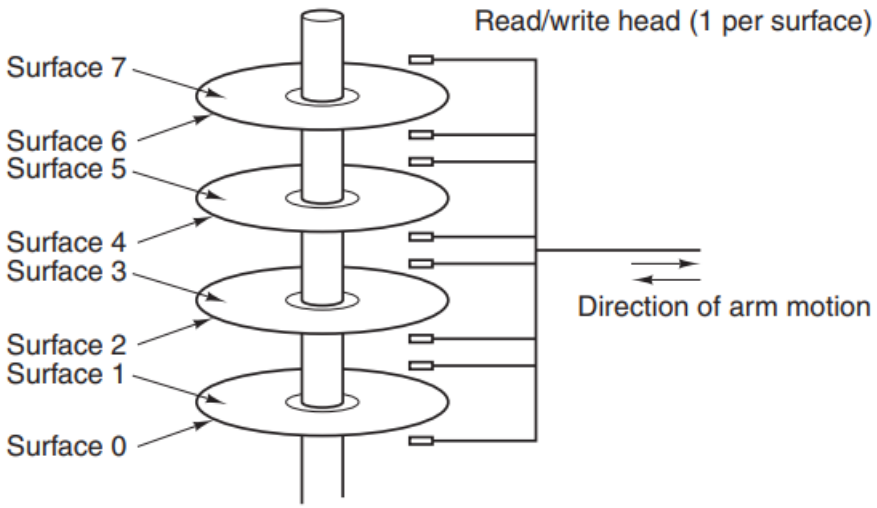

Fig. 1.10. Reprezentarea schematică a unui disc rigid.

Un hard disk este format din una sau mai multe plăci metalice care se rotesc la 5400, 7200, 10800 sau mai multe rotații pe minut. O unitate mecanică (capul de citire-acriere - CCS) se rotește la un anumit unghi deasupra acestor plăci, similar cu un vechi pick-up cu 33 de rotații pe minut. Informațiile sunt scrise pe disc într-o secvență de cercuri concentrice. La fiecare poziție de acționare dată, fiecare cap poate citi o secțiune circulară numită pistă. Un cilindru este format din combinația tuturor pistelor pentru o anumită poziție de acționare.

Fiecare pistă este împărțită într-un număr de sectoare, de obicei 512 octeți pe sector. La unitățile moderne, cilindrii exteriori conțin mai multe sectoare decât cilindrii interiori. Este nevoie de aproximativ 1 ms pentru a muta unitatea de la un cilindru la altul. Deplasarea la un cilindru selectat în mod arbitrar durează de obicei între 5 și 10 ms, în funcție de unitatea specifică. Atunci când CCP este poziționat deasupra pistei dorite, trebuie să se aștepte ca sectorul dorit să ajungă sub cap. Acest lucru are ca rezultat o altă întârziere de 5 până la 10ms în funcție de viteza de rotație a unității. Odată ce sectorul corespunzător se află sub capul unității, se va efectua o operațiune de citire sau de scriere, de la 50 MBytes/s (pentru unitățile cu viteză mai mică) până la 160 MB/s (pentru unitățile de mare viteză).

Uneori vorbim despre discuri care nu sunt cu adevărat discuri, cum ar fi discurile cu stare solidă (Solid State Disk - SSD). Acestea nu au părți mobile, nu au plăci metalice, iar datele sunt stocate în memoria flash. Sunt similare cu hard disk-urile și conțin o cantitate mare de date care nu se pierd atunci când se oprește alimentarea cu energie electrică.

Multe computere acceptă o schemă numită **memorie virtuală**. Acest aspect va fi abordat mai în detaliu în capitolul 3. Memoria virtuală vă permite să executați programe mai mari decât memoria fizică a computerului, plasând programul pe disc și folosind memoria RAM ca un fel de cache pentru părțile programului necesare la un moment de timp dat. Această schemă necesită o traducere transparentă a adresei generate de program în adresa fizică la care este stocat cuvântul în memoria RAM. Această mapare a adreselor este realizată de o parte a procesorului, numită unitatea de gestionare a memoriei (Memory Management Unit - MMU) sau managerul de memorie (figura 1.6).

Utilizarea memoriei cache și a MMU poate avea un impact semnificativ asupra performanței. În cazul lucrului cu multiprogramare, când se trece de la un program la altul, operație numită comutarea contextului, poate fi necesar să fie eliminate toate blocurile modificate din cache și modificate registrele de mapare din MMU. Ambele operațiuni sunt “foarte scumpe”, iar

programatorii încearcă din răsputeri să le evite. Unele consecințe ale tacticilor utilizate în acest scop vor fi analizate puțin mai târziu.

1.3.5 Dispozitivele de intrare/ieșire

Procesorul și memoria nu sunt singurele resurse pe care un sistem de operare trebuie să le gestioneze. Sistemul de operare interacționează în mod activ cu dispozitivele de intrare/ieșire. Figura 1.6 arată că dispozitivele de intrare/ieșire sunt de obicei alcătuite din două componente: dispozitivul propriu-zis și controlerul acestuia. Controlerul este un cip sau un set de cipuri care gestionează dispozitivul la nivel fizic. Acceptă comenzi de la sistemul de operare, cum ar fi citirea datele din dispozitiv și apoi le execută.

Destul de des, controlul direct al unui dispozitiv este foarte complex și necesită un nivel ridicat de detalii, astfel încât sarcina controlerului este de a oferi o interfață simplă (dar nu simplistă) cu sistemul de operare. De exemplu, un controler de disc poate primi o comandă de citire a sectorului 11206 de pe discul 2. La primirea comenzii, controlerul trebuie să convertească acest simplu număr secvențial de sector într-un număr de cilindru, sector și cap. Operațiunea de conversie poate fi complicată de faptul că cilindrii exteriori au mai multe sectoare decât cilindrii interiori, iar numerele rele de sector sunt alocate altor sectoare. Controlerul trebuie apoi să determine pe ce cilindru se află acum actuatorul capului și să emită o comandă pentru a-l deplasa înainte sau înapoi cu numărul necesar de cilindri. Apoi trebuie să aștepte până când sectorul corect ajunge sub cap și apoi să citească și să stocheze biții pe măsură ce aceștia sunt citiți, eliminând anteturile și calculând sumele de control. În cele din urmă, trebuie să asambleze biții în cuvinte și să le stocheze în memorie. Pentru a face toate acestea, controlerele includ adesea mici computere incorporate, programate să îndeplinească atfel de sarcini.

Cealaltă componentă este dispozitivul propriu zis. Dispozitivele au interfețe destul de simple, deoarece, în primul rând, au capacități foarte modeste și, în al doilea rând, trebuie să respecte standarde comune. Aceasta din urmă este necesar pentru a se asigura că, de exemplu, orice controler de disc SATA va funcționa cu orice disc SATA. SATA înseamnă Serial ATA, iar ATA înseamnă conexiune AT. În cazul în care nu știați ce înseamnă AT, acesta a fost derivat din a doua generație de calculatoare personale IBM cu tehnologie avansată (PC Advanced Technology), care se baza pe procesorul 80286, foarte puternic la acea vreme, cu o frecvență de 6 MHz și lansat de IBM în 1984. Din acest fapt, putem concluziona că industria informatică are obiceiul de a completa constant acronimele existente cu noi prefixe și sufixe. În plus, se poate concluziona că adjective precum "avansat" trebuie folosit cu grijă pentru a nu părea stupid peste 30 de ani.

SATA este tipul standard de unitate de disc pentru multe computere din zilele noastre. Deoarece interfața dispozitivului este ascunsă de către controlerul acestuia, toate sistemele de operare văd doar interfața controlerului, care poate fi foarte diferită de interfața dispozitivului.

Deoarece toate tipurile de controlere sunt diferite, pentru a le gestionae este nevoie software diferit. Software-ul conceput pentru a comunica cu controlerul, pentru a-i transmite comenzi și a primi răspunsuri de la acesta, se numește driver de dispozitiv. Fiecare producător de controlere trebuie să furnizeze drivere de dispozitiv pentru fiecare sistem de operare acceptat. De exemplu, scanerul poate fi livrat cu drivere pentru sistemele de operare OS X, Windows 7, Windows 8 și Linux, etc.

Pentru a fi utilizat, driverul trebuie să fie plasat în sistemul de operare, permițându-i astfel să ruleze în modul kernel. În general, driverele pot opera și în modul utilizator, iar suportul pentru modul kernel este oferit în prezent atât în sistemele de operare Linux, cât și Windows. Marea majoritate a driverelor sunt încă executate mai jos de granița nucleului. Driverele rulează în

modul utilizator doar pe foarte puține sisteme existente, cum ar fi MINIX 3. Driverele care sunt executate în mod utilizator trebuie să aibă permisiunea de a accesa dispozitivul într-un mod controlat, ceea ce reprezintă o sarcină departe de a fi simplă.

Există trei moduri de a instala un driver în kernel. Prima este de a recompila kernelul cu noul driver și apoi de a reporni sistemul. Multe sisteme UNIX vechi fac acest lucru. A doua metodă este de a crea o intrare într-un fișier special din sistemul de operare, informându-l despre ceea ce este necesar cu repornirea ulterioară a sistemului. Când sistemul de operare pornește, caută driverele de care are nevoie și le încarcă. Acesta este modul în care funcționează Windows. În cea de-a treia metodă - încărcarea dinamică a driverelor - sistemul de operare poate accepta drivere noi din mers și le poate instala rapid, fără a fi necesară o repornire. Această metodă era folosită anterior destul de rar, dar acum devine din ce în ce mai răspândită. Dispozitive externe conectabile la cald (hotpluggable)[4] cum ar fi dispozitivele USB și IEEE 1394, de exemplu, trebuie să aibă întotdeauna drivere care pot fi încărcate dinamic.

Fiecare controler are un număr mic de registre cu care trebuie să comunice. De exemplu, un controler de disc simplu poate avea registre pentru specificarea unei adrese pe unitate, adresa de memorie, contorul de sectoare și direcția de transfer a informațiilor (citire sau scriere). Pentru a activa un controler, driverul primește o comandă de la sistemul de operare, apoi o traduce în valorile corespunzătoare pentru a o scrie în registrele dispozitivului. Toate registrele dispozitivului se combină pentru a forma spațiul porturilor I/O, la care vom reveni în capitolul 5.

La unele calculatoare, registrele dispozitivului sunt mapate în spațiul de adrese al sistemului de operare (la acele adrese pe care acesta le poate utiliza), astfel încât stările registrelor pot fi citite și scrise în același mod ca și cuvintele obișnuite în memoria principală. Astfel de computere nu necesită nici un fel de comenzi I/O, iar programele de utilizator pot fi ținute în afara hardware prin plasarea acestor adrese în afara domeniului de acțiune al programelor (de ex. prin utilizarea registrelor de bază și a registrelor de delimitare a zonei de memorie). La alte calculatoare, registrele dispozitivelor sunt plasate într-un spațiu special de porturi I/O (I/O port space), unde fiecare registru are o adresă de port. Pe aceste mașini, există comenzi speciale de i/O executate în modul kernel (de obicei notate IN și OUT) pentru a permite driverelor să citească și să scrie date în registre. Prima schemă elimină necesitatea unor comenzi speciale de I/O, dar utilizează o parte din spațiul de adrese. Cea de-a doua schemă nu utilizează niciun spațiu de adrese, dar necesită instrucțiuni speciale. Ambele scheme sunt utilizate pe larg.

Introducerea și extragerea datelor se poate face în trei metodea diferite. În cea mai simplă dintre acestea, programul utilizatorului execută un apel de sistem care este tradus de kernel într-o procedură de apel de driver corespunzătoare. Driverul continuă apoi procesul de I/O. În acest timp, driverul execută un ciclu foarte scurt, interogând dispozitivul și monitorizând finalizarea operațiunii (de obicei, un bit special indică faptul că dispozitivul este ocupat). Când operațiunea de I/O este finalizată, driverul plasează datele (dacă există) acolo unde trebuie să ajungă și returnează controlul. Sistemul de operare returnează apoi controlul către programul apelant. Această metodă se numește așteptare activă sau așteptare a finalizării, iar dezavantajul este că procesorul este ocupat pe toată durata procesului de I/O.

În cadrul metodei a doua driverul lansează dispozitivul și îi solicită acestuia să-i trimită o întrerupere atunci când execuția comanzii este finalizată (intrarea sau ieșirea este încheiată). Imediat driverul returnează controlul. Sistemul de operare blochează programul apelant, dacă

[4] Adică, dispozitive care pot fi conectate sau deconectate de la calculator fără a fi nevoie să opriți sistemul de operare și să deconectați calculatorul de la sursa de alimentare. - Nota editorului.

este necesar, și trece la îndeplinirea altor sarcini. Când controlerul detectează sfârșitul transferului de date, generează o întrerupere pentru a semnala sfârșitul operațiunii.

Întreruperile joacă un rol foarte important în funcționarea sistemului de operare, așa că haideți să le analizăm mai îndeaproape. Figura 1.11 (a) prezintă pașii procesului de I/O. În cadrul pasului 1 driverul trimite o comandă către controler prin scrierea ei informații în registrele acestuia. Controlerul pornește apoi dispozitivul propriu-zis. La pasul 2, când controlerul termină de citit sau de scris un anumit număr de octeți, el setează la cipul de control al întreruperilor semnalul pentru microschema controlerului, folosind în acest scop anumite linii ale magistrașei. La pasul 3, dacă controlerul de întreruperi este pregătit să primească o întrerupere (și este întreruperi (este posibil să nu fie gata să primească întreruperea, dacă procesează o întrerupere cu o prioritate mai mare), acesta semnalizează pinul respectiv de pe cipul CPU informându-l că operațiunea s-a încheiat. În cel de-al patrulea pas, controlerul de întreruperi setează numărul dispozitivului pe magistrală, astfel încât procesorul să îl poată citi și să afle care dispozitiv a terminat numai ce lucrul (deoarece mai multe dispozitive pot funcționa în același timp).

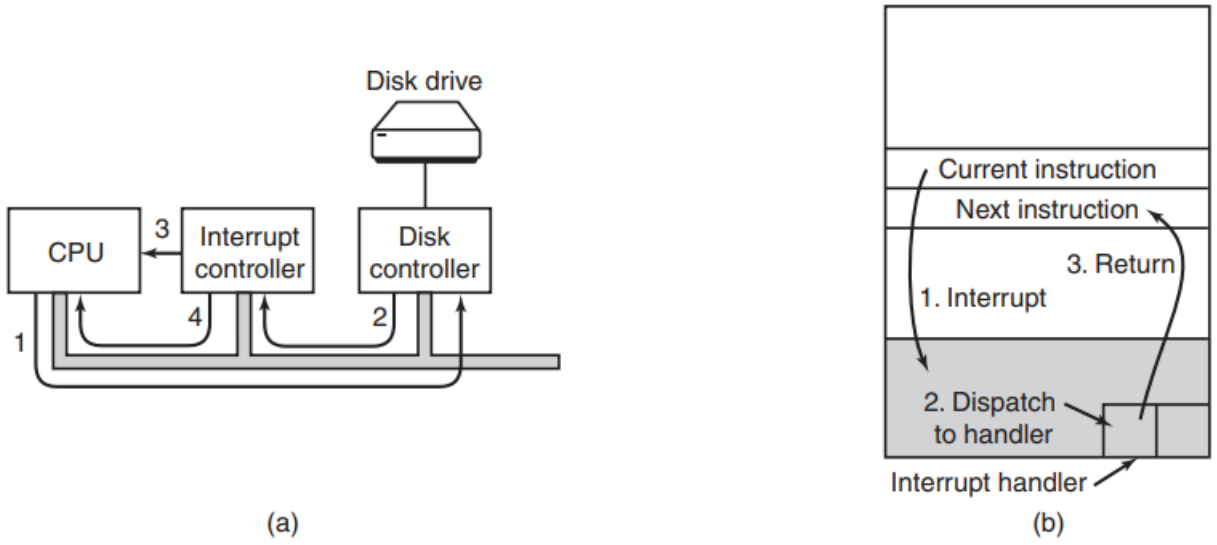

Fig. 1.11. a) Etapele de pornire a unui dispozitiv de I/O și de obținere a unei întreruperi. (b) Prelucrarea întreruperii implică preluarea întreruperii, rularea operatorului de întrerupere, și revenirea la programul utilizatorului.

De îndată ce procesorul decide să accepte o întrerupere, conținutul contorului de instrucțiuni și al cuvântului de stare a programului sunt plasate, de obicei, pe stiva curentă iar procesorul trece în modul kernel. Numărul dispozitivului poate fi folosit ca un index pentru porțiunea de memorie utilizată pentru a găsi adresa operatorului de întreruperi a dispozitivului respectiv. Această porțiune de memorie se numește vector de întrerupere. Atunci când un operator de întreruperi (care face parte din driverul dispozitivului, care emite cererea de întrerupere) își începe operațiunea, acesta extrage din stivă contorului de comenzi și cuvântul de stare a programului și le stochează, apoi interoghează dispozitivul pentru a determina starea acestuia. După tratarea întreruperii controlul estel returnat programului utilizatorului la instrucțiunea imediat următoare, care nu a fost încă executată (v. fig. 1.11 (b)).

Cea de-a treia metodă de intrare/ieșire utilizează un controler special de acces direct la memorie (DMA). Controlerul de acces direct la memorie este utilizat pentru a controla fluxul de biți între memoria principală și unele dintre controllere, fără a fi nevoie de o intervenție din partea unității centrale de procesare. Procesorul central configurează controlerul DMA

spunându-i câți octeți să transfere, ce dispozitiv și adresele care trebuie utilizate, în ce direcție trebuie transferate datele, și apoi îl lasă să funcționeze independent. Atunci când controlerul DMA termină lucrul, va emite o cerere de întrerupere, care este tratată în ordinea pe care am discutat-o anterior. Controlerul DMA și dispozitivele de intrare/ieșire vor fi prezentate mai detaliat în capitolul 5.

Întreruperile apar adesea în momente foarte nepotrivite, de exemplu în timpul lucrului unui operator al altei întreruperi. Din acest motiv, procesorul are capacitatea de a interzice tratarea unei întreruperi cu eliminarea ulterioară a interzicerii. În timp ce întreruperile sunt interzise, toate dispozitivele care și-au terminat activitatea vor continua să emită cereri de întrerupere, dar CPU nu se oprește din funcționare până când întreruperile nu vor fi permise din nou. Dacă mai multe dispozitive își finalizează solicitările de întrerupere în timpul perioadei de inhibare a întreruperilor, controlerul decide care din ele trebuie tratată prima, bazându-se de obicei pe priorități statice atribuite fiecărui dispozitiv. Dispozitivul cu cea mai mare prioritate este cel care câștigă. Toate celelalte dispozitive trebuie să își aștepte rândul.

1.3.6 Magistralele

Structura prezentată în figura 1.6 a fost utilizată timp de mulți ani în minicalculatoare și, de asemenea, în primul PC IBM. Dar, pe măsura sporirii vitezei de procesoare și a capacității memoriei posibilitățile magistralei unice (și, desigur, a mgistralei IBM PC) pentru a susține toate procesele de comunicare au fost epuizate. Trebuia de făcut ceva. Ca urmare, au apărut magistrale suplimentare, atât pentru dispozitive de intrare/ieșire mai rapide, cât și pentru schimbul de date între procesor și memorie. Ca o consecință a acestei evoluții, sistemul x86 produs în masă are în prezent forma prezentată în figura 1.12.

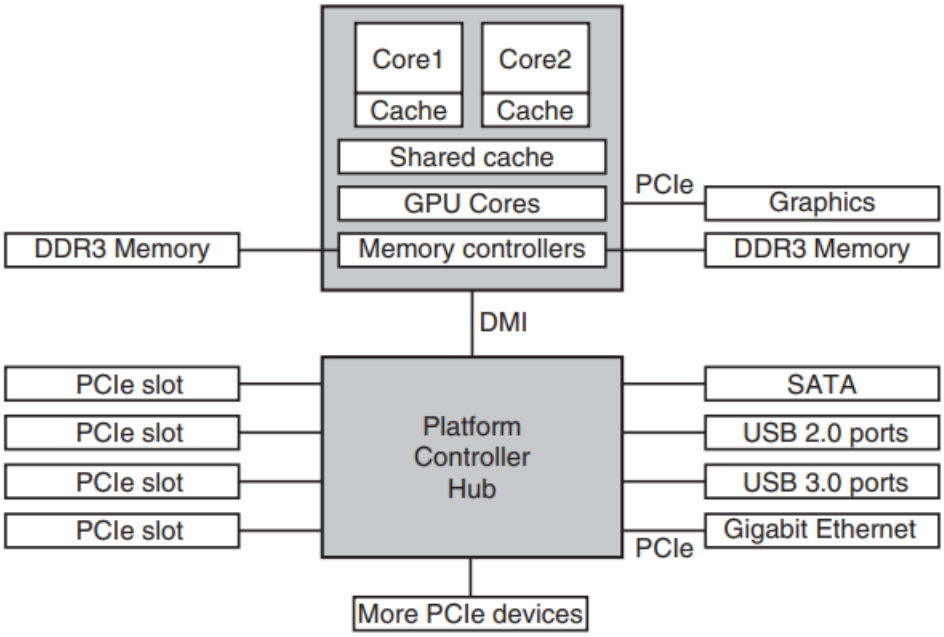

Fig. 1.12. Structura sistemului x86 extins.

Acest sistem are mai multe magistrale (de exemplu, magistrala cache, magistrala de memorie, de asemenea magistralele PCIe, PCI, USB, SATA și DMI), fiecare cu o viteză diferită de transfer de date și cu scop propriu. Pentru ca sistemul de operare să realizeze funcții de configurare și control, acesta trebuie să cunoască totul despre aceste magistrale. Magistrala

principală este magistrala PCI (Peripheral Component Interconnect interfața dispozitivelor periferice).

Magistrala PCIe a fost inventată de Intel ca succesor al magistralei mai vechi PCI, care, la rândul său, a înlocuit magistrala ISA (Industry Standard Architecture) originală. Deoarece poate transfera date la viteze de zeci de gigabiți pe secundă, magistrala PCIe funcționează mult mai rapid decât predecesoarea sa. Este, de asemenea, de natură foarte diferită. Până la introducerea sa în 2004, majoritatea magistralelor erau paralele și partajate. O arhitectură de magistrală partajată înseamnă că aceiași conductori sunt utilizați de diferite dispozitive pentru a transmite date. Prin urmare, atunci când mai multe dispozitive au date de transferat, este necesar un arbitru pentru a determina care dispozitiv va avea dreptul de a utiliza magistrala. În schimb, magistrala PCIe utilizează conexiuni directe dedicate punct-la-punct. O arhitectură de magistrală paralelă similară cu cea a PCI implică trimiterea fiecărui cuvânt de date pe mai mulți conductori. De exemplu, pe o magistrală PCI tipică, un singur număr de 32 de biți este trimis pe 32 de fire paralele. În schimb, PCIe utilizează o arhitectură de magistrală serială, iar toți biții unui mesaj sunt trimiși printr-un singur fir, cunoscut sub numele de pistă (lane), ceea ce este foarte asemănător cu trimiterea unui pachet de rețea. Acest lucru simplifică foarte mult sarcina, deoarece nu este nevoie să se asigure că toți cei 32 de biți ajung la destinație exact în același timp. Dar paralelismul este în continuare utilizat deoarece mai multe benzi pot funcționa în paralel. De exemplu, 32 de piste pot fi utilizate pentru a transmite 32 de mesaje în paralel. Datorită creșterii rapide a vitezelor de transfer de date ale dispozitivelor periferice, cum ar fi plăcile de rețea și adaptoarele grafice, standardul PCIe este actualizat la fiecare 3 - 5 ani. De exemplu, 16 benzi de PCIe 2.0 ofereau viteze de 64 Gbit/s. Un upgrade la PCIe 3.0 dublează această viteză, iar un upgrade la PCIe 4.0 ar dubla-o din nou.

În același timp, există încă multe dispozitive tradiționale pentru standardul PCIe mai vechi. După cum se poate observa din figura 1.12, aceste dispozitive sunt conectate la un hub separat. În viitor, când PCI nu va mai fi considerat doar o magistrală veche, ci una străveche, va fi posibil ca toate dispozitivele PCI să fie conectate la un alt hub care, la rândul său, le va conecta la hub-ul principal, creând astfel un arbore al magistralelor.

În această configurație, procesorul comunică cu memoria prin intermediul unei magistrale rapide DDR3, cu dispozitivul grafic extern prin intermediul unei magistrale PCIe și cu toate celelalte dispozitive prin intermediul concentratorului prin intermediul magistralei DMI (Direct Media Interface). La rândul său, hub-ul conectează toate celelalte dispozitive folosind o magistrală serială universală pentru comunicarea cu dispozitivele USB, o magistrală SATA pentru comunicarea cu hard disk-urile și unitățile DVD și o magistrală PCIe pentru transferul de cadre Ethernet. Dispozitivele PCI vechi care utilizează o magistrală PCI tradițională au fost deja menționate aici.

Magistrala USB (Universal Serial Bus) a fost concepută pentru a conecta la computer toate dispozitivele de intrare/ieșire de viteză redusă, cum ar fi tastaturile și mouse-ul. Dar nu ar fi normal să numim "lent" un dispozitiv USB 3.0 de 5Gb/s într-o generație în care computerul IBM PC era considerat a fi un computer de masă cu magistrala ISA de 8 Mbits/s. USB utilizează un un conector mic cu (în funcție de versiune) 4 - 11 pini, dintre care unii alimentează dispozitivele USB sau sunt conectați la masă.

USB este o magistrală centralizată în care dispozitivul principal (rădăcină) interoghează dispozitivele de I/O la fiecare milisecundă pentru a vedea dacă acestea au date de transferat. Standardul USB 1.0 putea oferi o rată de transfer agregată de 12 Mbps, USB 2.0 a crescut la 480 Mbps, iar USB 3.0 a atins un maxim de cel puțin 5 Gbps. Un dispozitiv USB putea fi

conectat la calculator și putea funcționa imediat, fără a fi nevoie de repornire, necesară pentru unele dispozitive înainte de USB, ceea ce îngrozea o generație întreagă de utilizatori frustrați.

SCSI (Small Computer System Interface) este o magistrală de mare viteză concepută pentru unități de înaltă performanță, scannere și alte dispozitive care necesită o lățime de bandă mare. În prezent, aceste magistrale sunt utilizate în principal la servere și stații de lucru. Se pot atinge viteze de transfer de date de până la 640 MB/s.

Pentru a funcționa în mediul prezentat în figura 1.12, sistemul de operare trebuie să știe ce periferice sunt conectate la computer și să configureze aceste dispozitive. Această cerință i-a determinat pe Intel și Microsoft să dezvolte un sistem pentru computerele PC compatibile, numit plug and play. Acesta se bazează pe un concept similar implementat inițial în Apple Macintosh. Înainte de plug and play, fiecare placă I/O avea un nivel fix de solicitare a întreruperilor și adrese fixe pentru registrele lor de I/O. De exemplu, tastatura utiliza întreruperea 1 și adresele de I/O de la 0x60 la 0x64; controlerul de dischetă utiliza întreruperea 6 și adresele de I/O de la 0x3F0 la 0x3F7; imprimanta utiliza întreruperea 7 și adresele de I/O de la 0x378 la 0x37A etc.

Toate acestea au funcționat foarte bine o perioadă. Problemele au început atunci când utilizatorul cumpăra o plachetă de sunet și un modem intern și constata că ambele dispozitive foloseau, să zicem, întreruperea 4. Apărea un conflict care împiedica dispozitivele să lucreze împreună. Soluția a fost apariția comutatoarelor DIP sau jumperi pe fiecare placă de I/O. Însă, utilizatorul trebuia să fie instruit să selecteze nivelurile de solicitare a întreruperilor și adresele de intrare/ieșire pentru dispozitiv care să nu intre în conflict cu toate celelalte întreruperi și adrese utilizate în sistemul său. Uneori, adolescenții care și-au dedicat viața rezolvării puzzle-urilor legate de hardware-ul calculatoarelor reușeau să îndeplinească aceste cerințe fără erori. Dar, din nefericire, aproape nimeni în afară de ei nu a putut face acest lucru, ceea ce a dus la un haos total.

Tehnologia Plug and play forțează sistemul să colecteze automat informații despre dispozitivele de I/O, atribuind în mod centralizat niveluri de solicitare a întreruperilor și adrese I/O, iar apoi spunând fiecărei plăci ce valori îi sunt atribuite. Această acțiune este strâns legată de pornirea calculatorului și ar trebui să examinăm acest proces, deoarece nu este atât de simplu pe cât pare la prima vedere.

1.3.7 Pornirea computerului

Succint, un computer pornește după cum urmează. Fiecare calculator personal are o placă de bază (care în SUA acum, ca urmare a răspândirii corectitudinii politice în industria informatică, se numește placă de bază)[5]. Pe placa de bază se află un program, numit BIOS (Basic Input Output System). BIOS-ul conține software de nivel scăzut pentru I/O, inclusiv proceduri pentru citirea stării tastaturii, de afișare a informațiilor pe ecran și de execuție a operațiilor de I/O cu discul. În zilele noastre acest software este stocat într-o memorie flash nevolatilă cu acces aleatoriu care poate fi actualizată. BIOS este actualizat de către sistemul de operare în cazul în care sunt detectate diverse erori.

La pornirea inițială a computerului, BIOS-ul primul începe lucrul. Acesta verifică mai întâi cantitatea de memorie RAM instalată pe computer și prezența tastaturii, precum și instalarea și răspunsul normal al altor dispozitive importante. Totul începe cu scanarea magistralelor PCIe și PCI pentru a identifica dispozitivele conectate la acestea. Unele dintre aceste dispozitive sunt moștenite din trecut (adică dezvoltate înainte ca tehnologia plug and play să fie creată).

[5] Este, de asemenea, denumită, în mod corect, placa de sistem sau placa principală. - Nota editorului.

Acestea au niveluri de întrerupere și adrese de I/O fixe (eventual setate prin comutatoare sau jumperi pe placa I/O, dar care nu pot fi modificate de sistemul de operare). Aceste dispozitive sunt înregistrate. De asemenea, sunt înregistrate și dispozitivele compatibile cu Plug and play. În cazul în care dispozitivele prezente sunt diferite de cele înregistrate în sistem la ultima pornire, sunt configurate noile dispozitive.

În continuare, BIOS-ul va determina dispozitivul de pe care va fi realizată încărcarea, prin verificarea incrementală a dispozitivelor din lista salvată în memoria CMOS. Utilizatorul poate face modificări la această listă intrând în programul de configurare BIOS imediat după pornirea inițială. De obicei, se încearcă o pornire de pe un CD-ROM (uneori de pe un stick USB), dacă sistemul conține unul. În caz de eșec, sistemul pornește de pe hard disk. Primul sector este citit în memorie de pe dispozitivul de pornire și apoi este executat programul stocat în el. De obicei, acest program verifică tabelul de partiții, care se află la sfârșitul sectorului de pornire, pentru a determina care partiție are statutul activ. Un încărcător de boot secundar este apoi citit de pe această partiție, care la rândul său citește de pe partiția activă și pornește sistemul de operare.

Sistemul de operare solicită apoi BIOS-ului informații despre configurația computerului. Se verifică dacă există drivere pentru fiecare dispozitiv. Dacă careva nu este disponibil, sistemul de operare vă solicită să instalați CD-ul cu driverul (furnizat de producătorul dispozitivului) sau descarcă driverul de pe site-ul Internet. Odată ce are în posesie toate driverele de dispozitiv, sistemul de operare le încarcă în kernel. Apoi își inițializează propriile tabele, creează toate procesele de fundal de care are nevoie și rulează programul de logare sau interfață grafică cu utilizatorul.

1.4 Gradina zoologica a sistemelor de operare

Istoria sistemelor de operare datează de mai bine de o jumătate de secol. În acest timp, a fost dezvoltată o mare varietate de sisteme de operare, dar nu toate au cunoscut o răspândire largă. În această secțiune, vom examina pe scurt nouă sisteme de operare. Vom reveni asupra unora dintre aceste diferite tipuri de sisteme de operare în paginile acestei cărți.

1.4.1 Sisteme de operare pentru masinile mari de calcul

Cea mai înaltă categorie include sistemele de operare pentru mainframe-uri – calculatoare mari, care ocupă săli întregi și care se găsesc încă în centrele de date ale marilor întreprinderi. Acestea diferă de calculatoarele personale prin cantitatea de date introduse și extrase. Mainframe-urile cu mii de discuri și petabytes de date sunt foarte obișnuite, iar un computer personal cu un astfel de arsenal ar fi invidiat de toată lumea. Mainframe-urile sunt, de asemenea, utilizate ca servere web puternice, servere pentru magazine online de mari dimensiuni și servere pentru tranzacții între companii.

Sistemele de operare pentru mainframe-uri sunt concepute în principal pentru a gestiona mai multe sarcini simultan, majoritatea necesitând cantități enorme de date de intrare/ieșire. Acestea oferă de obicei trei tipuri de servicii: procesare pe loturi, procesare de tranzacții și partajare a timpului. Procesarea pe loturi este un sistem de procesare a lucrărilor standard fără intervenția utilizatorului. Prelucrarea pe loturi este procesarea cererilor de despăgubire ale companiilor de asigurări sau a rapoartelor de vânzări ale lanțului de magazine. Sistemele de procesare a tranzacțiilor gestionează un număr mare de cereri mici, cum ar fi procesarea cecurilor bancare sau rezervarea biletelor de avion. Fiecare tranzacție elementară este mică în volum, dar sistemul poate gestiona sute sau mii de tranzacții pe secundă. Lucrul în regim de partajare a timpului face posibilă pentru mai mulți utilizatori la distanță să își execute simultan sarcinile pe un computer, cum ar fi interogarea simultană a unei baze de date mar. Toate

aceste funcții sunt strâns legate între ele și adesea sistemele de operare ale mașinilor universale le execută pe toate împreună. Un exemplu de sistem de operare pentru mașini universale este OS/390, succesorul lui OS/360. Însă, aceste sisteme de operare sunt treptat înlocuite cu variante de sisteme de operare UNIX, de exemplu Linux.

1.4.2 Sisteme de operare pentru servere

La un nivel un pic mai jos se află sistemele de operare pentru servere. Acestea rulează pe servere, care sunt computere personale foarte puternice, stații de lucru sau chiar mașini universale. Acestea deservesc simultan prin rețea mai mulți utilizatori, oferindu-le acces comun la resurse hardware și software. Serverele pot furniza servicii de imprimare, stocare de fișiere sau servicii web. De obicei, furnizorii de servicii Internet operează mai multe servere pentru a-și servi clienții. La deservirea site-urilor web, serverele stochează paginile web și gestionează cererile primite. Sisteme de operare tipice pentru servere sunt Solaris, FreeBSD, Linux și Windows Server 201x

1.4.3 Sisteme de operare multiprocesor

Astăi din ce în ce mai mult este utilizată combinarea mai multor procesoare centrale într-un singur sistem pentru a obține o putere de calcul demnă de marile ligi. În funcție de modul exact în care are loc această agregare și de resursele partajate, aceste sisteme se numesc computere paralele, multicomputere sau sisteme multiprocesor. Acestea necesită sisteme de operare speciale, care sunt adesea versiuni ale sistemelor de operare pentru servere, echipate cu funcții speciale de comunicare, interfațare și sincronizare.

Odată cu apariția recentă a procesoarelor multi-core pentru calculatoarele personale, chiar și sistemele de operare convenționale pentru desktop-uri și laptop-uri au început să ruleze cel puțin un mic sistem multiprocesor. În timp, se pare că numărul de nuclee nu va face decât să crească. Din fericire, de-a lungul anilor de cercetare anterioară s-au acumulat cunoștințe vaste despre sistemele de operare multiprocesor, iar utilizarea acestui arsenal în sistemele multi-core nu ar trebui să cauzeze prea multe complicații. Partea cea mai dificilă va fi găsirea de aplicații care să poată exploata toată această putere de calcul. Multe sisteme de operare populare, inclusiv Windows și Linux, pot rula pe sisteme multiprocesor.

1.4.4 Sisteme de operare pentru calculatoare personale

Următoarea categorie include sistemele de operare pentru calculatoare personale. Toți reprezentanții lor moderni acceptă multitaskingul. Destul de des, zeci de programe rulează simultan încă din timpul procesului de pornire. Sarcina sistemelor de operare pentru calculatoarele personale este de a sprijini activitatea utilizatorului individual într-un mod calitativ. Acestea sunt utilizate pe scară largă pentru procesarea de texte, crearea de foi de calcul, jocuri și accesarea Internetului. Exemple tipice sunt Linux, FreeBSD, Windows 7, Windows 8 și OS X de la Apple. Sistemele de operare pentru calculatoare personale sunt atât de cunoscute încât nu au nevoie de o prezentare specială. De fapt, multe oamenii nu sunt conștienți că există și alte tipuri de sisteme de operare.

1.4.5 Sisteme de operare în timp real

Continuând să coborâm spre sisteme din ce în ce mai simple, am ajuns acum la tablete, smartphone-uri și alte calculatoare portabile. Aceste computere, cunoscute inițial sub numele de PDA (Personal Digital Assistant), sunt calculatoare de mici dimensiuni care se țin în mână în timpul lucrului. Cele mai cunoscute dintre acestea sunt smartphone-urile și tabletele. După cum

s-a menționat deja, această piață este dominată de sistemele de operare Android de la Google și iOS de la Apple, dar acestea au mulți concurenți. Cele mai multe dintre aceste dispozitive dispun de procesoare multi-core, GPS, camere și alți senzori, memorie amplă și sisteme de operare sofisticate. În plus, toate acestea au mai multe aplicații terțe (apps) pentru memorii USB decât vă puteți imagina.

1.4.6 Sisteme de operare încorporate

Sistemele încorporate rulează pe calculatoare care controlează diverse dispozitive. Deoarece aceste sisteme nu sunt concepute pentru a avea programe de utilizator instalate, ele nu sunt considerate de obicei computere. Printre exemplele de dispozitive în care sunt instalate calculatoare încorporate se numără cuptoarele cu microunde, televizoarele, mașinile, inscriptoarele DVD, telefoanele convenționale și playerele MP3. În general, sistemele încorporate se deosebesc prin faptul că, în nici un caz, pee le nu vor putea fi lansate aplicații de la terți. Nu este posibil să descărcați o nouă aplicație pe un cuptor cu microunde, deoarece toate programele sale sunt stocate în memoria ROM. În consecință, nu este nevoie să protejăm aplicațiile una de cealaltă, iar sistemul de operare poate fi simplificat. Embedded Linux, QNX și VxWorks sunt considerate a fi cele mai populare sisteme de operare din acest domeniu.

1.4.7 Sisteme de operare pentru nodurile de senzori

Rețelele alcătuite din noduri senzoriale miniaturale conectate între ele și la stația de bază prin canale fără fir sunt utilizate în diverse scopuri. Astfel de rețele de senzori sunt utilizate pentru protejarea perimetrelor clădirilor, securizarea frontierelor naționale, detectarea incendiilor de pădure, măsurarea temperaturii și a nivelului precipitațiilor pentru prognoza meteo, colectarea de informații despre mișcările inamicului pe câmpul de luptă și multe altele.

Nodurile din această rețea sunt computere miniaturale alimentate cu baterii, cu un sistem radio încorporat. Acestea au o putere limitată și trebuie să funcționeze timp îndelungat în mod nesupravegheat în aer liber, adesea în condiții climatice dificile. Rețeaua trebuie să fie suficient de fiabilă și permită defecțiuni ale componentelor individuale, care, pe măsură ce bateriile își pierd capacitatea vor apărea din ce în ce mai des.

Fiecare nod de senzori este un computer real dotat cu un procesor, memorie RAM și memorie permanentă și unul sau mai mulți senzori. Acesta rulează un sistem de operare mic, dar real, de obicei bazat pe evenimente și care răspunde la evenimente externe sau face măsurători periodice pe baza semnalelor ceasului încorporat. Sistemul de operare ar trebui să fie mic și simplu, deoarece principalele probleme ale acestor noduri sunt capacitatea redusă a memoriei RAM și autonomia limitată a bateriei. Ca și în cazul sistemelor incorporate, toate programele sunt preîncărcate și utilizatorii nu pot rula un program descărcat din Internet, simplificând astfel întregul proiect. Un exemplu bine cunoscut un exemplu de sistem de operare pentru noduri de sensori este TinyOS.

1.4.8 Sisteme de operare de timp real

O altă varietate de sisteme de operare sunt sistemele de timp real. Aceste sisteme se caracterizează prin faptul că aici timpul este un parametru cheie. De exemplu, în sistemele de control al proceselor industriale, calculatoarele în timp real trebuie să colecteze informații despre procese și să le utilizeze pentru a controla mașinile din fabrică. Destul de des, acestea trebuie să îndeplinească cerințe foarte stricte în materie de sincronizare. De exemplu, atunci când o mașină se deplasează de-a lungul unei linii de asamblare, trebuie să aibă loc operațiuni specifice în anumite momente. Dacă un robot de sudură începe să sudeze prea devreme sau

prea târziu, utilajul va fi nefuncțional. În cazul în care operațiunea trebuie efectuată exact la timp (sau la un anumit moment în timp), sistemul este unul de timp real. Multe astfel de sisteme se regăsesc în controlul proceselor de producție, în aviație și în industria aerospațială, în domeniul militar și în alte aplicații similare. Acestea trebuie să ofere asigurări absolute că anumite acțiuni vor fi efectuate la un moment dat.

Un alt tip de astfel de sisteme este sistemul **soft real-time**, în care, deși nedorit, este permis ca o acțiune să nu cauzeze un prejudiciu ireparabil prin nerespectarea unui termen limită. Această categorie include sistemele audio sau multimedia digitale. Telefoanele inteligente sunt, de asemenea, sisteme soft real-time.

Deoarece cerințele pentru sistemele în timp real sunt foarte stricte, sistemele de operare sunt uneori simple biblioteci interfațate cu programe de aplicații, în care totul este strâns interconectat și nu există nicio protecție reciprocă. Un exemplu de astfel de sistem ar fi eCos.

Categoriile de sisteme de operare pentru PDA-uri, sistemele incorporate și sistemele de timp real se suprapun în mare măsură în ceea ce privește caracteristicile lor intrinseci. Aproape toate acestea au cel puțin unele aspecte ale sistemelor soft real-time. Sistemele încorporate și sistemele în timp real funcționează numai cu software-ul introdus în ele de către dezvoltatorii acestor sisteme; utilizatorii nu pot adăuga propriul software la acest arsenal, ceea ce face mult mai ușoară protejarea lor. PDA-urile și sistemele încorporate sunt concepute pentru consumatorii individuali, în timp ce sistemele în timp real sunt mai des utilizate în producția industrială. Cu toate acestea, în ciuda tuturor acestor lucruri, ele au în comun o serie de asemănări.

1.4.9 Sisteme de operare pentru carduri inteligente

Cele mai mici sisteme de operare rulează pe carduri inteligente. Cardul inteligent este un dispozitiv de mărimea unui card de credit cu procesor propriu. Sistemele de operare pentru acestea sunt supuse unor limitări foarte stricte în ceea ce privește puterea de procesare a procesorului și volumul memoriei. Unele dintre cardurile inteligente sunt alimentate de contactele cititorului în care sunt introduse, în timp ce altele, cardurile inteligente fără contact, sunt alimentate prin inducție, ceea ce le limitează semnificativ capacitățile. Unele pot face față unei singure funcții, cum ar fi plata electronică, dar există și carduri inteligente multifuncționale. Acestea sunt adesea sunt adesea sisteme proprietare.

Unele carduri inteligente sunt concepute pentru a utiliza limbajul Java. Acest lucru înseamnă că ROM-ul unui card inteligent conține un interpretor Java Virtual Machine (JVM). Applet-urile Java (programe mici) sunt încărcate pe card și sunt executate de către interpretorul JVM. Unele dintre aceste carduri sunt capabile să gestioneze mai multe applet-uri Java în același timp, ceea ce presupune lucrul în modul multiprogram și necesitatea de a prioritiza executarea programelor. Atunci când două sau mai multe applet-uri sunt executate în același timp, devin relevante aspectele de gestionare a resurselor și de securitate care trebuie rerzolvate de către sistemul de operare disponibil pe card (de obicei destul de primitiv).

1.5 Concepte din domeniul sistemelor de operare

Majoritatea sistemelor de operare folosesc anumite concepte de bază și abstractizări, precum procese, spații de adrese și fișiere, care joacă un rol major în înțelegerea sistemelor. În următoarele secțiuni vom analiza succint, așa cum ar trebui să fie într-o introducere, câteva dintre aceste concepte de bază. Vom reveni la detaliile fiecărui concept în capitolele următoare. Pentru a ilustra aceste concepte, din când în când vom folosi exemple luate în principal din

UNIX. De regulă, exemple similare pot fi găsite și în alte sisteme de operare, iar unele dintre ele vor fi luate în considerare ulterior.

1.5.1 Procese

Conceptul cheie în toate sistemele de operare este procesul. Un proces este în esență un program în timpul executării sale. Fiecare proces are spațiul de adrese asociat - o listă de adrese de celule de memorie de la zero la un anumit maxim, de unde procesul poate citi date și unde le poate scrie. Spațiul de adrese conține programul executabil, datele programului și stiva. În plus, fiecare proces are asociat un set de resurse, care, de obicei, include regiștri (inclusiv un contor de comenzi sau contorul ordinal și un indicator de stivă), o listă de fișiere deschise, diferite semnale - avertizări neprocesate, o listă de procese asociate și toate celelalte informații necesare în timpul execuției programului. Astfel, un proces este un container care conține toate informațiile necesare pentru ca programul să funcționeze.

Conceptul unui proces va fi discutat mai detaliat în Capitolul 2, iar acum, pentru a dezvolta o idee intuitivă a procesului, considerăm un sistem care funcționează în regim multiprogram. Utilizatorul poate porni programul de editare video și specifica conversia fișierului video cu durata de o oră în orice format specific (procesul va dura câteva ore), apoi poate trece la navigarea prin Internet. În același timp, un proces de fundal poate începe să funcționeze, care se „trezește” periodic pentru a verifica e-mailurile primite. Vom avea astfel (cel puțin) trei procese active: un editor video, un browser web și un program de e-mail (client). Periodic, sistemul de operare va lua decizii de a opri un proces și de a începe altul, posibil datorită faptului că primul și-a epuizat cota de timp de procesorului în secunda anterioară.

Dacă procesul este suspendat în acest fel, ulterior ar trebui să fie continuat din starea în care a fost oprit. Aceasta înseamnă că în timpul perioadei de suspendare, toate informațiile despre proces trebuie stocate explicit undeva. De exemplu, un proces poate avea mai multe fișiere deschise simultan pentru citire. Un indicator pentru poziția curentă este asociat cu fiecare dintre aceste fișiere (adică numărul de octeți sau înregistrări care urmează să fie citite în continuare). Când procesul se întrerupe, toate aceste indicii trebuie salvate astfel încât apelul *read*, care va fi executat după reluarea procesului, să conducă la citirea corectă a datelor necesare. În multe sisteme de operare, toate informațiile despre fiecare proces, cu excepția conținutului propriului spațiu de adrese, sunt stocate într-un tabel al sistemului de operare, care se numește **tabelul proceselor** și este un tablou (sau o listă legată) de structuri, una pentru fiecare dintre procesele existente la fiecare moment de timp dat.

Astfel, procesul (inclusiv unul suspendat) constă din spațiul propriu de adrese, denumit în mod obișnuit **imaginea în memorie**, și intrările din tabelul proceselor cu conținutul regiștrilor săi, precum și alte informații necesare pentru reluarea ulterioară a procesului.

Principalele apeluri de sistem utilizate în controlul proceselor sunt cele asociate cu crearea și terminarea proceselor. De exemplu, procesul numit **interpretorul comenzilor**, sau **shell**, citește comenzile de la terminal. Dacă utilizatorul introduce o comandă de compilare a unui program, shell-ul va crea un nou proces care pornește compilatorul. Când acest proces finalizează compilarea, va efectua un apel de sistem pentru a-și încheia existența.

Dacă un proces este capabil să creeze alte procese (numite procese descendent), iar aceste procese, la rândul lor, pot crea propriile procese, relația respectivă poate fi reprezentată printr-un arbore de procese similar cu cel dinn fig. 1-13.

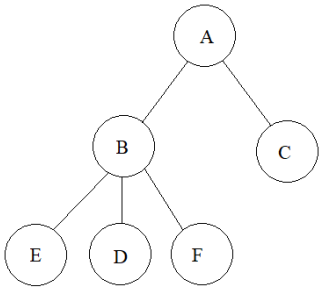

Fig. 1-13. Arborele proceselor

Procesele conexe care lucrează împreună pentru a finaliza o sarcină trebuie adesea să facă schimb de date între ele și să-și sincronizeze acțiunile. Această relație se numește comunicare între procese și va fi discutată detaliat în Capitolul 2.

Alte apeluri de sistem concepute pentru a controla procesul permit solicitarea alocării de memorie suplimentară (sau eliberarea memoriei inactive), organizarea așteptării finalizării procesului descendent sau încărcarea unui alt program peste acesta.

Uneori, este necesară transmiterea unor informații către un proces lansat anterior, care nu este în stare de așteptare a acestor informații. Un exemplu este un proces care comunică cu un alt proces care rulează pe un alt computer și trimite un mesaj către procesul de la distanță din rețea. Pentru a se asigura contra unei posibile pierderi a mesajului sau a răspunsului la mesaj, expeditorul poate solicita sistemul său de operare să-l anunțe după un anumit interval de timp care-i situația, astfel încât să poată retrimite mesajul dacă nu primește confirmarea primirii acestuia mai devreme. După setarea unui astfel de cronometru, programul poate continua să facă alte lucrări.

La expirarea intervalului setat, sistemul de operare trimite un semnal alarmă la proces. Acest semnal forțează procesul să întrerupă lucrarea în desfășurare, să salveze starea regiștrilor săi și să înceapă o procedură specială pentru procesarea alarmei, pentru a transmite repetat, de exemplu, mesajul presupus pierdut. Când manipulatorul de semnal își va termina activitatea, procesul întrerupt își va relua execuția din aceeași stare în care a fost înainte de sosirea semnalului. Semnalele sunt analogi software cu întreruperile hardware. Ele pot fi generate în diverse situații și nu numai după ora stabilită în cronometru. Multe întreruperi hardware, cum ar fi încercarea de a executa o instrucțiune interzisă sau de a accesa o adresă interzisă, sunt transmise procesului care a generat eroarea.

Fiecare utilizator căruia i se permite să lucreze cu sistemul i se atribuie un **ID de identificare a utilizatorului** (User Identification (UID)) de către administratorul de sistem. Fiecare proces are UID-ul utilizatorului care l-a lansat. Procesele descendent au același UID ca și procesul părinte. Utilizatorii pot face parte dintr-un grup, fiecare grup având propriul identificator (Group Identification (GID)).

Un utilizator cu o valoare UID specială, numită superuser pe UNIX și administrator pe Windows, are privilegii speciale care pot să încalce multe reguli de securitate. În sistemele de calcul mari, numai administratorul de sistem cunoaște parola necesară pentru obținerea drepturilor de utilizator, dar mulți utilizatori obișnuiți (în special studenții) depun eforturi considerabile pentru a găsi vulnerabilitățile sistemului care le-ar permite să devină superuser fără parolă.

1.5.2 Spații de adrese

Fiecare computer are o anumită cantitate de memorie operativă folosită de programele executabile. În cele mai simple sisteme de operare, există un singur program în memorie.

Pentru a începe un alt program, mai întâi trebuie eliminat programul curent după care să fie încărcat celălalt program în locul primului.

Sistemele de operare mai sofisticate permit încărcarea mai multor programe în memoria centrală. Pentru a elimina interferențele reciproce (și interferența cu funcționarea sistemului de operare), este necesar un mecanism de protecție. Chiar dacă acest mecanism face parte din echipament, el este controlat de sistemul de operare. Acest punct de vedere este legat de problemele de gestionare și protecție a memoriei RAM. O altă problemă de memorie, dar nu mai puțin importantă, este gestionarea spațiului de adrese al procesului. Fiecărui proces îi este atribuit pentru utilizare un set continuu de celule de memorie cu adrese, de obicei de la zero la maxim. În cel mai simplu caz, volumul maxim de locațiuni alocate unui proces este mai mic decât capacitatea memoriei operative. Astfel, procesul își poate umple spațiul de adrese și există suficient spațiu pentru plasarea sa în memoria RAM.

Cu toate acestea, multe computere folosesc adrese de 32 sau 64 biți, ceea ce permite să avem un spațiu de adrese de 2^{32}, respectiv 2^{64} octeți. Ce se întâmplă dacă spațiul de adrese al procesului depășește capacitatea RAM a calculatorului, iar procesul trebuie să utilizeze întregul său spațiu? În primele calculatoare acest lucru era imposibil. Astăzi există tehnologia, numită memorie virtuală, în care sistemul de operare stochează unele fragmente ale spațiului de adrese în RAM și altele pe disc, schimbându-și fragmentele în funcție de necesități. În esență, sistemul de operare creează o abstractizare a spațiului de adrese ca un set de adrese la care se poate referi procesul. Spațiul de adrese este separat de memoria fizică a mașinii și poate fi mai mare decât încape în memoria fizică. Gestionarea spațiilor de adrese și a memoriei fizice este o parte importantă a sistemului de operare.

1.5.3 Fișierele

Un alt concept cheie susținut de aproape toate sistemele de operare este sistemul de fișiere. După cum a fost menționat anterior, funcția principală a sistemului de operare este de a ascunde specificul discurilor și al altor dispozitive de intrare / ieșire și de a oferi programatorului un model abstract convenabil și simplu de înțeles, format din fișiere independente de dispozitiv. Evident, apelurile de sistem sunt necesare pentru a crea, șterge, citi și scrie fișiere. Înainte ca fișierul să fie gata de citire, acesta trebuie să fie găsit pe disc și deschis, iar după citire, trebuie să fie închis. Pentru aceste operațiuni, sunt furnizate apeluri de sistem.

Pentru a oferi un loc pentru stocarea fișierelor, multe sisteme de operare pentru calculatoarele personale folosesc directorul ca o modalitate de a combina fișierele în grupuri. De exemplu, un student poate avea un director pentru fiecare curs studiat (pentru programele necesare pentru acest curs), un alt director pentru e-mail și un al treilea pentru pagina sa de pornire. Pentru crearea și ștergerea directoarelor sunt necesare apeluri de sistem. Sunt necesare operații de introducere sau de ștergere a unui fișier existent în/din director. În calitate de elementele ale unui director pot fi fișierele sau alte directoare. Acest model a devenit prototipul structurii ierarhice a sistemului de fișiere, una dintre variante fiind prezentată în fig. 1-14.

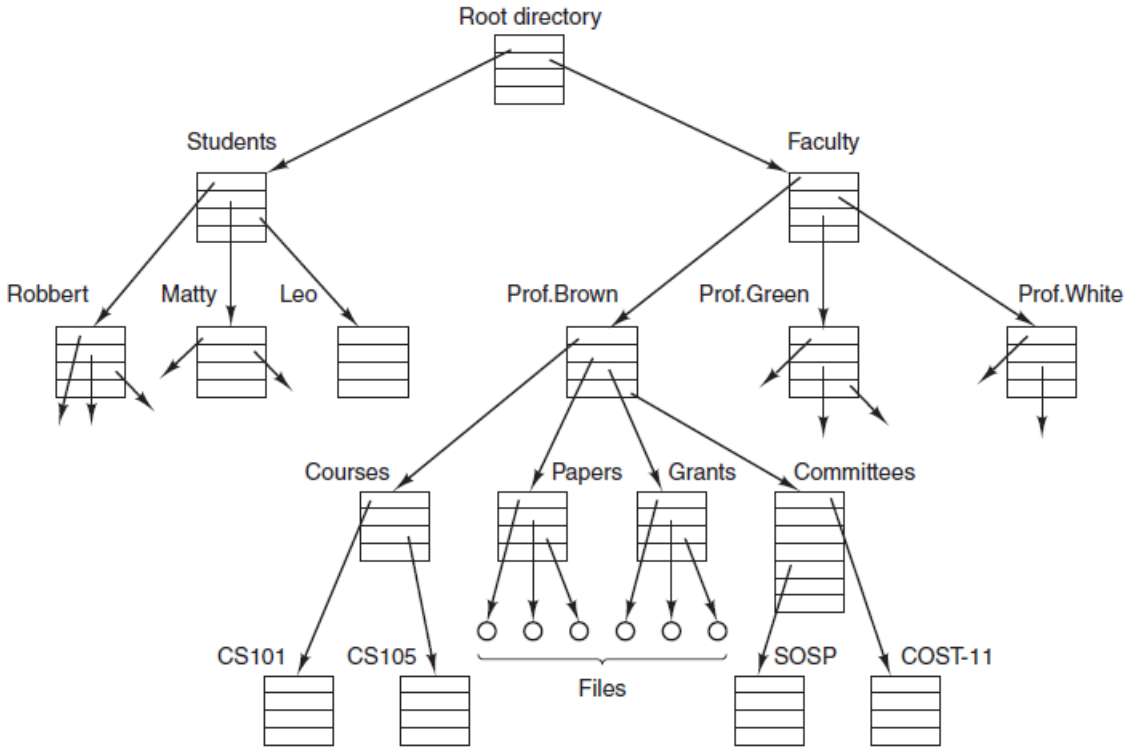

Fig. 1-14. Sistemul de fișiere pentru un departament universitar

Ambele ierarhii de procese și fișiere sunt organizate în arbori, dar asemănarea se oprește aici. Ierarhiile de proces nu sunt de obicei foarte profunde (mai mult de trei niveluri întâlnindu-se rar), în timp ce ierarhiile de fișiere sunt pe patru, cinci sau chiar mai multe niveluri. Ierarhiile de proces sunt de obicei de scurtă durată, minute cel mult, în timp ce ierarhia de directoare poate exista ani de zile. Proprietatea și protecția diferă, de asemenea, pentru procese și fișiere. De obicei, doar un proces părinte poate controla sau chiar accesa un proces copil, dar aproape întotdeauna există mecanisme care permit ca fișierele și directoarele să fie citite de un grup mai larg decât doar de proprietar.

Fiecare fișier din ierarhia de directoare poate fi specificat folosind **numele de cale** (path name) - calea care începe de la directorul rădăcină până la numele fișierului. Astfel de nume absolute de cale sunt formate din lista de directoare care trebuie parcurse pornind de la directorul rădăcină pentru a ajunge la fișier, cu bare oblice care separă componentele. În fig. 1-14, calea pentru fișierul CS101 este /Faculty/Prof.Brown/Courses/CS101. Prima bară oblică indică faptul că calea este absolută, adică începe de la directorul rădăcină. În Windows, caracterul “\” (bară oblică inversă (backslash) este utilizat ca separator în loc de caracterul „/” (din motive istorice), astfel că calea dată mai sus ar fi scrisă în Windows ca \Faculty\Prof.Brown\Cursuri\CS101. În toată această carte vom folosi în general convenția UNIX pentru căi.

La fiecare moment de timp orice proces are un **director de lucru** (director curent), în care sunt folosite c[i care nu încep cu o bară. De exemplu, în fig. 1-14, dacă /Faculty/Prof.Brown ar fi directorul de lucru, utilizarea căii Cursuri/CS101 ar conduce la același fișier ca și calea absolută de mai sus. Procesele își pot schimba directorul de lucru prin emiterea unui apel de sistem care specifică noul director de lucru. Pentru a putea fi citit sau scris un fișier trebuie mai întâi deschis, moment în care sunt verificate permisiunile. Dacă accesul este permis, sistemul returnează un număr întreg numit **descriptor de fișier** pentru a fi utilizat în operațiunile ulterioare. Dacă accesul este interzis, este returnat un cod de eroare.

Un alt concept important în UNIX este sistemul de fișiere montat. Majoritatea computerelor desktop au una sau mai multe unități optice în care pot fi inserate CD-ROM-uri, DVD-uri sau discuri Blu-ray. Acestea au aproape întotdeauna porturi USB, în care pot fi conectate stick-urile de memorie USB (adevărate unități de disc solid), iar unele computere au dischete sau hard diskuri externe. Pentru a oferi un mod elegant de a face față acestor suporturi amovibile UNIX permite ca sistemul de fișiere de pe discul optic să fie atașat la arborele principal. Fie situația din fig. 1-15 (a). Înainte de apelul de montare, sistemul de fișiere rădăcină, de pe hard disk și un al doilea sistem de fișiere, pe un CD-ROM, sunt separate și nu au nici o legătură.

Sistemul de fișiere de pe CD-ROM nu poate fi utilizat, deoarece nu există nicio modalitate de a specifica numele căilor pentru fișierele de pe acesta. UNIX nu permite ca numele de cale să fie prefixate cu un nume sau număr de unitate; aceasta ar fi exact genul de dependență de dispozitiv pe care sistemele de operare ar trebui să-l ocolească. Apelul de sistem montare permite atașarea sistemului de fișiere de pe CD-ROM la sistemul de fișiere rădăcină.

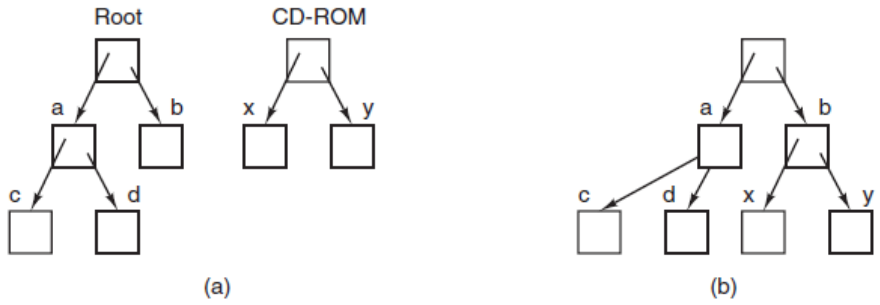

Fig. 1-15. (a) Înainte de montare fișierele de pe CD-ROM nu sunt accesibile. (b) După montare ele devin parte a ierarhiei de fișiere

În Fig. 1-15 (b) sistemul de fișiere de pe CD-ROM a fost montat pe directorul b, permițând astfel accesul la fișiere /b/x și /b/y. Dacă directorul b ar fi conținut fișiere, acestea nu ar fi accesibile în timp ce CD-ROM-ul a fost montat, deoarece /b s-ar referi la directorul rădăcină al CD-ROM-ului. (A nu fi capabil să accesezi aceste fișiere nu este la fel de grav pe cât pare la început: sistemele de fișiere sunt aproape întotdeauna montate pe directoare goale.) Dacă un sistem conține mai multe discuri dure, acestea pot fi montate și într-un singur arbore.

Un alt concept important în UNIX este conceptul de fișier special. Fișierele speciale sunt furnizate pentru ca dispozitivele de intrare/ieșire să pară fișiere. În acest fel, ele pot fi citite și scrise folosind aceleași apeluri de sistem ca și cele utilizate pentru citirea și scrierea fișierelor. Există două tipuri de fișiere speciale: fișiere speciale orientate pe blocuri și fișiere speciale orientate pe caractere. Fișierele speciale bloc orientate sunt utilizate pentru modelarea dispozitivelor care constau dintr-o colecție de blocuri adresabile aleatoriu, cum ar fi discurile. La deschiderea unui fișier special bloc orientat care citește, să zicem, blocul 4, un program poate accesa direct al patrulea bloc de pe dispozitiv, fără a ține cont de structura sistemului de fișiere. În mod similar, fișierele speciale orientate pe caractere sunt utilizate pentru modelarea imprimantelor, modemurilor și a altor dispozitive care acceptă sau produc un flux de caractere. Prin convenție, fișierele speciale sunt păstrate în directorul /dev. De exemplu, /dev/lp ar putea fi imprimanta (odată numită imprimantă linie). Ultima noțiune despre care vom discuta aici se referă atât la procese, cât și la fișiere este noțiunea de **conductă** (pipe). O conductă este un fel de pseudofișier care este utilizat pentru a conecta două procese (fig. 1-16).

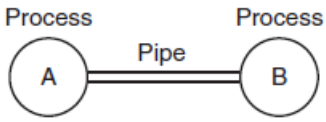

Fig. 1-16. Două procese conectate prin intermediul unei conducte (pipe)

Dacă procesele A și B doresc să discute folosind o conductă, acestea trebuie să o configureze din timp. Când procesul A dorește să trimită date pentru procesul B, acesta scrie pe conductă ca și cum ar fi un fișier de ieșire. De fapt, implementarea unei conducte seamănă foarte mult cu cea a unui fișier. Procesul B poate citi datele citind din conductă ca și cum ar fi un fișier de intrare. Astfel, comunicarea între procesele din UNIX seamănă foarte mult ca un fișier obișnuit în care un proces scrie, iar altul citește. Mai mult chiar: singurul mod prin care un proces poate descoperi că fișierul de ieșire în care scrie nu este un fișier adevărat, ci o conductă, este folosirea unui apel sistem special. Sistemele de fișiere sunt foarte importante. Vom avea mult mai multe de spus despre ele în cap. 4 și, de asemenea, în capitolele 10 și 11.

1.5.4 Intrări/Ieșiri

Toate calculatoarele au dispozitive fizice de intrare și de ieșire. Există multe tipuri de dispozitive de intrare și ieșire, inclusiv tastaturi, monitoare, imprimante etc. Este în sarcina sistemului de operare să gestioneze aceste dispozitive. În acest scop orice sistem de operare are un subsistem de intrare/ieșire (I/E) pentru gestionarea dispozitivelor sale. O parte din software-ul I/E este independent de dispozitiv, adică este pentru mai multe sau chiar pentru toate dispozitivele de intrare/ieșire. Alte părți ale acestui soft, cum ar fi driverele de dispozitiv, sunt specifice anumitor dispozitive de I / O. În cap. 5 vom analiza software-ul I/E.

1.5.5 Protecția și securitatea

Calculatoarele conțin cantități mari de informații pe care utilizatorii doresc adesea să le protejeze și să le păstreze confidențial. Aceste informații pot include mesaje de e-mail, planuri de afaceri, declarații fiscale și multe altele. Sistemul de operare trebuie să asigure securitatea sistemului, astfel încât fișierele, de exemplu, să fie accesibile numai utilizatorilor autorizați.

Un exemplu simplu din UNIX: în UNIX fișierele sunt protejate atribuind fiecăruia un cod de protecție binară pe 9 biți. Codul de protecție este format din trei câmpuri pe 3 biți, unul pentru proprietar, unul pentru ceilalți membri ai grupului proprietarului (utilizatorii sunt împărțiți în grupuri de către administratorul de sistem) și unul pentru toți ceilalți. Fiecare câmp are un bit pentru acces la citire, un bit pentru acces la scriere și un bit pentru acces la executare. Acești 3 biți sunt cunoscuți sub numele de biți **rwx**. De exemplu, codul de protecție rwxr-x - x înseamnă că proprietarul poate citi, scrie sau executa fișierul, alți membri ai grupului pot citi sau executa fișierul (dar nu pot scrie) și toți ceilalți pot doar executa (dar nu pot citi sau scrie) fișierul. Pentru un director, x indică permisiunea de căutare. O liniuță înseamnă că permisiunea respectivă nu există.

Pe lângă protecția fișierelor, există multe alte probleme de securitate. Protejarea sistemului împotriva intrusilor nedoriți, atât umani, cât și non-umani (de exemplu, viruși) este una dintre ele. Vom analiza diverse probleme de securitate din cap. 9.

1.5.6 Interpretorul de comenzi (shell)

Sistemul de operare este codul care efectuează apelurile de sistem. Editoarele, compilatoarele, asambloarele, utilitarele sau interpretorul limbajului de comandă cu siguranță nu fac parte din sistemul de operare, chiar dacă sunt importante și utile. Cu riscul de a confunda oarecum

lucrurile, în această secțiune vom analiza pe scurt interpretorul limbajului de comandă UNIX - **shell**-ul. Deși nu face parte din sistemul de operare, acesta folosește în mod intensiv multe funcții ale sistemului de operare și, astfel, servește ca un bun exemplu al modului în care sunt utilizate apelurile de sistem. De asemenea, este interfața principală între un utilizator care stă la terminalul său și sistemul de operare, cu excepția cazului în care utilizatorul utilizează o interfață grafică cu utilizatorul. Există multe shell-uri, inclusiv sh, csh, ksh și bash. Toate acceptă funcționalitatea descrisă mai jos, care derivă din shell-ul original (sh).

Când un utilizator se conectează, un shell este pornit. Acesta are terminalul ca intrare și ieșire standard. Începe afișând invitația la lucru (promptul), un caracter cum ar fi un semn dolar, care îi spune utilizatorului că shell-ul așteaptă introducerea unei comenzi. Dacă utilizatorul introduce

date

de exemplu, shell-ul creează un proces copil pentru a rula programul *date*. La terminare, shell-ul afișează din nou promptul și așteaptă următoarea comandă.

Utilizatorul poate specifica faptul că ieșirea standard va fi redirecționată către un fișier, de exemplu,

date > *file*

În mod similar, intrarea standard poate fi redirecționată, ca în

sort <*file*1>*file*2

care va lansa programul de sortare *sort* cu intrarea preluată din *file*1 și ieșire trimisă la *file*2.

Ieșirea unui program poate fi utilizată ca intrare pentru un alt program, conectându-le cu o conductă. Prin urmare prin introducerea liniei

*cat file*1 *file*2 *file*3 | *sort* >/dev/lp

este lansat programul *cat* care va concatena trei fișiere și va trimite ieșirea programului *sort* care va aranja toate liniile în ordine alfabetică. Rezultatul sortării este redirecționat către fișierul special /dev/lp, care este de obicei imprimanta.

Dacă utilizatorul adaugă simbolul ampersand (&) după o comandă, shell-ul nu așteaptă să fie terminată sortarea și afișează imediat promptul. Prin urmare,

*cat file*1 *file*2 *file*3 | *sort* >/dev/lp &

pornește *sort* ca o lucrare de fundal, permițând utilizatorului să continue să lucreze normal în timp ce sortarea continuă. Shell-ul are o serie de alte caracteristici interesante, pe care nu avem spațiu pentru a le discuta aici.

Majoritatea calculatoarelor personale în aceste zile folosesc interfața grafică (GUI). De fapt, GUI este doar un program care rulează deasupra sistemului de operare, precum un shell. În sistemele Linux, acest fapt este evident, deoarece utilizatorul are de ales din (cel puțin) două interfețe grafice: Gnome și KDE sau deloc (folosind o fereastră de terminal pe X11). În Windows, este de asemenea posibil să înlocuiți desktop-ul GUI standard (Windows Explorer) cu un program diferit, schimbând unele valori din registru, deși puțini oameni fac acest lucru.

1.6 APELURI DE SISTEM

Am văzut că sistemele de operare au două funcții principale: furnizarea de abstractizări pentru programatori și programele utilizatorilor și gestionarea resurselor computerului. În cea mai mare parte, interacțiunea dintre programele utilizatorului și sistemul de operare se referă la prima

funcție, de exemplu, crearea, scrierea, citirea și ștergerea fișierelor. Partea de gestionare a resurselor este în mare măsură transparentă pentru utilizatori și realizată automat. Astfel, interfața dintre programele utilizatorului și sistemul de operare se referă în primul rând la abordarea abstractizărilor. Pentru a înțelege cu adevărat ce fac sistemele de operare, trebuie să examinăm îndeaproape această funcție de intermediar. Apelurile de sistem utilizate în acest caz variază de la un sistem de operare la altul (deși conceptele de bază tind să fie similare).

Suntem astfel obligați să facem o alegere între (1) generalități vagi (" sistemele de operare au apeluri de sistem pentru citirea fișierelor ") și (2) unele sisteme specifice (“UNIX are un apel de sistem *read* cu trei parametri: unul pentru specificarea fișierului, al doilea spune unde vor fi puse datele și al treilea stabilește câți octeți vor fi citiți").

Noi am ales ultima abordare. Vom avea mai mult de lucru mergând pe acesată cale, dar putem afla mai multe informații despre ceea ce fac cu adevărat sistemele de operare. Deși această discuție se referă în mod special la POSIX (Standardul Internațional 9945-1), deci și la UNIX, System V, BSD, Linux, MINIX 3 și așa mai departe, majoritatea celorlalte sisteme de operare moderne au apeluri de sistem care îndeplinesc aceleași funcții, chiar dacă detaliile diferă. Deoarece mecanica efectivă a emiterii unui apel de sistem depinde în mare măsură de mașină și deseori trebuie exprimată în cod de asamblare, o bibliotecă de proceduri este oferită pentru a face posibilă efectuarea de apeluri de sistem din programe C și deseori și din alte limbi.

Este util să aveți în vedere următoarele. Orice computer cu un singur procesor poate executa o singură instrucțiune la un moment de timp dat. Dacă un proces rulează un program de utilizator în modul utilizator și are nevoie de un serviciu de sistem, cum ar fi citirea datelor dintr-un fișier, acesta trebuie să execute o instrucțiune specială (TRAP) pentru a transfera controlul sistemului de operare. Sistemul de operare află despre ce dorește procesul apelant prin inspectarea parametrilor. Apoi execută apelul de sistem și returnează controlul procesului la instrucțiunea imediat următoare apelului de sistem. Într-un anumit sens, a face un apel de sistem este ca și cum ai face un tip special de apel de procedură, doar că apelurile de sistem intră în kernel, pe când apelurile de procedură nu pot face acest lucru.

Pentru a clarifica mecanismul apelurilor de sistem, să aruncăm o privire rapidă la apelul de sistem *read*. Așa cum am menționat mai sus, acesta are trei parametri: primul care specifică fișierul din care vor fi citiți niște octeți, cel de-al doilea stabilește buferul folosit pentru citire, iar al treilea indică câți octeți vor fi citiți. Ca aproape toate apelurile de sistem, *read*e ste invocat din programele C, apelând la o procedură de bibliotecă cu același nume ca apelul de sistem: *read*. Un apel de sistem într-un program C poate arăta astfel:

count = *read* (*fd*, *buffer*, *nbytes*);

Apelul de sistem (și procedura de bibliotecă) returnează numărul de octeți citiți efectiv în *count*. Această valoare este în mod normal aceeași cu *nbytes*, dar poate fi mai mică, dacă, de exemplu, în timpul lecturii este întâlnit sfârșitul fișierului.

Dacă apelul de sistem nu poate fi efectuat din cauza unui parametru nevalid sau a unei erori de disc, *count* este setat în -1 și numărul erorii este introdus într-o variabilă globală, *errno*. Programele ar trebui să verifice întotdeauna rezultatele unui apel de sistem pentru a vedea dacă a apărut o eroare.

Apelurile de sistem sunt efectuate într-o serie de pași. Pentru a clarifica acest concept, să examinăm apelul *read* discutat mai sus. Pregătindu-se pentru apelarea procedurii de bibliotecă *read*, care efectiv va efectua apelul de sistem *read*, programul apelant introduce mai întâi parametrii în stiva de execuție, așa cum se poate observa în etapele 1-3 din fig. 1-17.

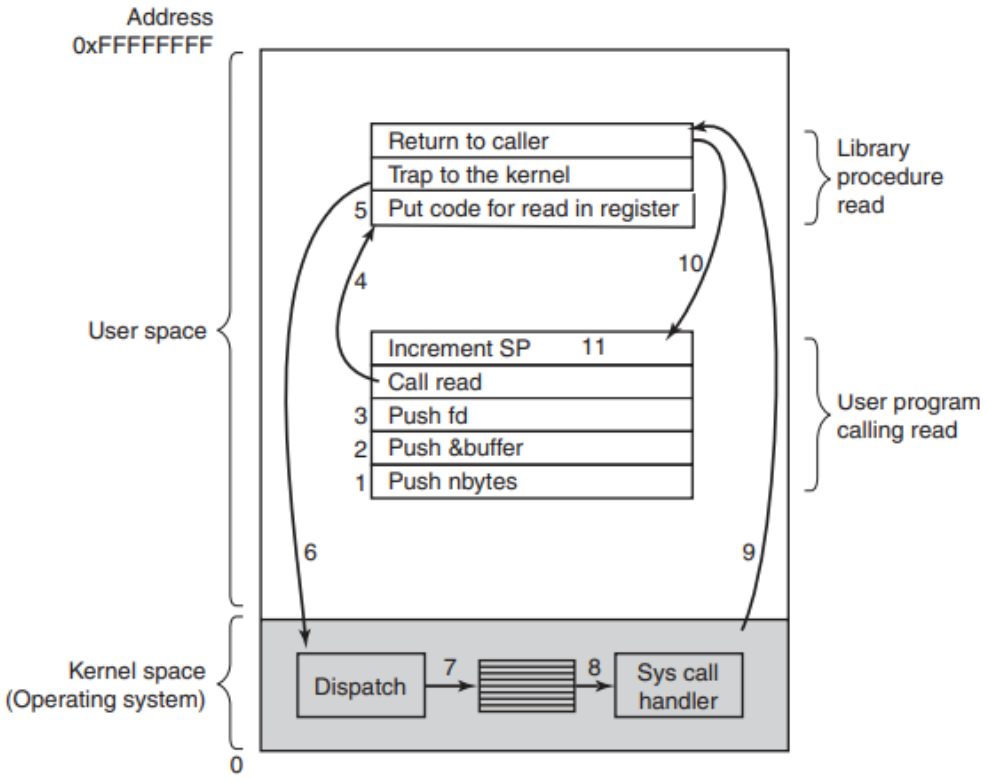

Fig. 1.17. Cei 11 pași în realizarea apelului de sistem read(fd, buffer, nbytes).

Compilatoarele C și C ++ introduc parametrii în stivă în ordine inversă din motive istorice. Primul și al treilea parametru sunt desemnați prin valoare, iar al doilea este desemnat prin referință, ceea ce înseamnă că adresa buferului (indicată de &) este folosită, nu conținutul acestuia. Apoi are loc apelul efectiv la procedura bibliotecii (pasul 4). Această este o instrucțiune obișnuită de apel de procedură, folosită pentru apelarea tuturor procedurilor.

Procedura de bibliotecă, posibil scrisă în limbajul de asamblare, plasează de obicei numărul apelului de sistem într-un loc în care sistemul de operare îl așteaptă, cum ar fi un registru (pasul 5). Apoi, execută o instrucțiune TRAP pentru a trece de la modul utilizator la modul kernel și începe executarea la o adresă fixă din kernel (pasul 6). Instrucțiunea TRAP este de fapt destul de similară cu instrucțiunea de apelare a procedurii, doar că instrucțiunea care urmează este preluată dintr-o locație la care are acces doar sistemul de operare, iar adresa de retur este salvată pe stivă pentru a fi utilizată ulterior.

Instrucțiunea TRAP diferă, de asemenea, de instrucțiunea de apel procedură în două moduri fundamentale. În primul rând, asigură trecerea în modul kernel. O instrucțiune obișnuită de apel de procedură nu face acest lucru. În al doilea rând, în loc să dea o adresă relativă sau absolută unde se află procedura, instrucțiunea TRAP nu poate trece la o adresă arbitrară. În funcție de arhitectură, fie sare la o singură locație fixă, fie că există un câmp de 8 biți în instrucțiune - indexul în tabelul din memorie care conține adrese de salt sau echivalente.

Codul kernel-ului care începe în urma TRAP-ului examinează numărul apelului de sistem și apoi este trimis către handler-ul apelurilor de sistem, de obicei via un tabel de indicatoare către manipulatoarele de apel sistem indexate pe numărul de apel de sistem (pasul 7). La pasul 8 este executat codul funcției solicitate a sistemului de operare. După ce-și încheie execuția, controlul poate fi returnat procedurii de bibliotecă în spațiul utilizatorului la instrucțiunea

următoare instrucțiunii TRAP (pasul 9). Această procedură returnează apoi controlul programului de utilizator în modul obișnuit (pasul 10).

Pentru a termina lucrarea, programul utilizator trebuie să curețe stiva, așa cum se întâmplă după orice apel de procedură (pasul 11). Presupunând că stiva crește în jos, așa cum se întâmplă adesea, codul compilat crește indicatorul stivei suficient de exact pentru a elimina parametrii introduși în stivă înainte de apelul *read*. Programul va rula în continuare conmform instrudțiunilor sale.

În pasul 9 de mai sus, am spus că „controlul poate fi returnat procedurii de bibliotecă în spațiul utilizatorului”, pentru un motiv întemeiat. Apelul de sistem poate bloca apelantul, împiedicându-l să continue. De exemplu, dacă încearcă să citească de pe tastatură și nu s-a tastat nimic, apelantul trebuie blocat. În acest caz, sistemul de operare va stabili dacă nu cumva poate fi rulat un alt proces. Mai târziu, când este disponibilă intrarea dorită, acest proces va atrage atenția sistemului și va parcurge pașii 9–11.

În secțiunile următoare, vom examina unele dintre cele mai utilizate apeluri de sistem POSIX, sau mai precis, procedurile de bibliotecă care efectuează aceste apeluri de sistem. POSIX are aproximativ 100 de apeluri de procedură. Unele dintre cele mai importante sunt enumerate în Fig. 1-18, grupate pentru comoditate în patru categorii.

În mare măsură, serviciile oferite de aceste apeluri determină cea mai mare parte a activității sistemului de operare, deoarece gestionarea resurselor pe computerele personale este minimă (cel puțin în comparație cu mașinile mari cu mai mulți utilizatori). Serviciile includ lucruri precum crearea și încheierea proceselor, crearea, ștergerea, citirea și scrierea fișierelor, gestionarea directoarelor și efectuarea intrării și ieșirilor.

1.6.1 Apeluri de sistem pentru administrarea proceselor

Primul grup de apeluri din Fig. 1-18 se ocupă de gestionarea proceselor.Este corect să începem cu *fork*, care este singura modalitate de a crea un nou proces în POSIX. Această funcție un duplicat exact al procesului părinte, inclusiv descriptorii de fișiere, registrele sunt la fel. După execuția lui *fork*, procesul inițial și copia (părintele și copilul) merg pe căi separate. Toate variabilele au valori identice în momentul *fork*, dar din moment ce datele părintelui sunt copiate pentru a crea copilul, modificările ulterioare ale uneia dintre ele nu o afectează pe cealaltă. (Textul programului, care nu poate fi modificat, este împărțit între părinte și copil.) Apelul *fork* returnează o valoare, care este zero pentru procesul copil și egală cu PID (ID-ul de proces) copilului la părinte. Folosind PID-ul returnat, cele două procese pot vedea care este procesul părinte și care este procesul copil.

În cele mai multe cazuri, după un *fork*, copilul va trebui să execute un cod diferit față de părinte. Fie ca exemplu cazul shell-ului. Este citită o comandă de la terminal, prin *fork* este creat un proces copil, așteaptă ca să fie executată comanda (proceul copil), la încheierea procesului copil este citită următoarea comandă. Pentru a aștepta finalizarea proceului copil, părintele execută un apel de sistem *waitpid*, care îl face să aștepte terminarea procesului copil (orice copil dacă există mai mult de unul). Apelul lui *waitpid* poate conduce la așteptarea unui anumit proces copil sau oricărui proces copil, setând primul parametru la −1. Când *waitpid* se încheie, adresa indicată de al doilea parametru, *statloc*, va fi setată la starea de ieșire a procesului copil (valoarea normală sau anormală de ieșire). De asemenea, sunt oferite diferite opțiuni, specificate de al treilea parametru. De exemplu, întoarcerea imediată dacă niciun copil nu a ieșit deja.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Fig. 1-18. Câteva dintre cele mai importante apeluri de sistem POSIX

Să vedem cum *fork* este folosit de shell. Când o comandă este tastată, shell-ul crează un nou proces. Acest proces copil trebuie să execute comanda utilizatorului. Face acest lucru folosind apelul de sistem *execve*, ceea ce face ca întreaga sa imagine de bază să fie înlocuită cu fișierul numit în primul său parametru. (De fapt, apelul de sistem propriu-zis este *exec*, dar mai multe proceduri de bibliotecă îl apelează cu parametri diferiți și nume ușor diferite. Vom trata acesta ca apeluri de sistem aici). Un shell extrem de simplificat, care ilustrează utilizarea lui *fork*, *waitpid*, și a *execve* este arătată în fig. 1-19.

În cel mai general caz, *execve* are trei parametri: numele fișierului care urmează să fie lansat în execuție, un pointer către tabloul de argumente și un pointer către tabloul de mediu. Există diverse rutine de bibliotecă, inclusiv *execl*, *execv*, *execle* și *execve*, pentru a permite omiterea sau specificarea parametrilor în diferite moduri. Noi vom folosi *exec* pentru a reprezenta apelul de sistem invocat de toate acestea.

Să luăm în considerare cazul unei comenzi cum ar fi

*cp file*1 *file*2

utilizat pentru a copia *file*1 în *file*2. După ce shell-ul a fost solicitat, procesul copil localizează și execută fișierul *cp* și îi transmite numele fișierelor sursă și țintă.

```
#define TRUE 1
while (TRUE) {                          /*ciclu infinit*/
   type_promt();                        /*afișarea promt-ului*/
   read_command(command, parameters); /*citește de la terminal*/
   if(fork()!=0) {                      /*a fost creat un process copil*/
     /*Codul procesului părinte */
     Waitpid(-1, &status, 0);           /*a;teaptă terminarea pr.copil*/
   } else {
     /*Codul procesului copil */
     Execve(command, parameters, 0);  /*comanda de executare*/
   }
}
```

Fig. 1-19. Cum funcționează un Shell

Programul principal al *cp* (și programul principal al majorității celorlalte programe C) conține declarația

main(*argc*, *argv*, *envp*)

unde *argc* este numărul de elemente din linia de comandă, inclusiv numele programului. Pentru exemplul de mai sus, *argc* este 3.

Al doilea parametru, *argv*, este un pointer către un tablou. Elementul *i* al acestui tablou este un pointer către șirul cu numărul *i* din linia de comandă. În exemplul nostru, *argv*[0] punctează pe șirul „*cp*”, *argv*[1] pe șirul „*file*1”, iar *argv*[2] pe „*file*2”.

Al treilea parametru al lui *main*, *envp*, este un pointer către mediul de execuție, un tablou de șiruri care conține atribuiri de forma *name* = *value* folosite pentru a transmite programelor informații precum tipul terminalului și numele directorului principal. Există proceduri de bibliotecă la care programele pot apela pentru a obține variabile de mediu, care sunt adesea utilizate pentru a personaliza modul în care un utilizator dorește să îndeplinească anumite sarcini (de exemplu, imprimanta implicită pe care să o utilizeze). În fig. 1-19, niciun mediu nu este trecut copilului, deci al treilea parametru al *execve* este 0.

Dacă *exec* vi se pare complicat, nu disperați; este (semantic) cel mai complicat dintre toate apelurile de sistem POSIX. Toate celelalte sunt mult mai simple. De exemplu, apelul *exit*, utilizat de procese atunci când își termină execuția. Are un singur parametru, *starea de ieșire* (de la 0 la 255), care este returnat părintelui prin *statloc* în apelul sistem *waitpid*.

Spațiul de adrese al proceselor UNIX este împărțit în trei segmente: segmentul text (codul programului), segmentul de date (variabilele) și segmentul de stivă. Segmentul de date crește în sus și stiva crește în jos, așa cum vedem în fig. 1-20.

Între ele este spațiul de adrese neutilizat (gap). Stiva crește automat în gap, conform necesităților, iar extinderea segmentului de date se face explicit folosind un apel de sistem, *brk*, care specifică noua adresă unde urmează să se termine segmentul de date. Acest apel nu este definit de standardul POSIX, deoarece programatorii sunt încurajați să utilizeze procedura de bibliotecă *malloc* pentru alocarea dinamică a spațiului de stocare. Însă implementarea de bază a *malloc* nu a fost creată a fi un subiect potrivit pentru standardizare, deoarece puțini programatori o folosesc direct și este îndoielnic că oricine observă că *brk* nu se află în POSIX.

1.6.2 Apeluri de sistem pentru administrarea fisierelor

Multe apeluri de sistem se referă la gestiunea fișierelor. În această secțiune vom analiza apelurile care operează pe fișiere individuale; în următoarea, vom examina cele care implică directoare sau sistemul de fișiere în ansamblu.

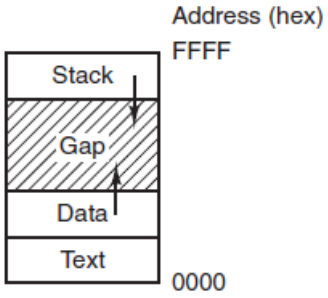

Fig. 1-20. Spațiul de adrese al unui proces

Pentru citire sau scriere fișierul trebuie mai întâi deschis. Acest apel specifică numele fișierului care trebuie deschis, fie folosind calea absolută sau în raport cu directorul curent, precum și modul preconizat de lucru *O_RDONLY*, *O_WRONLY* sau *O_RDWR*, adică deschis pentru citire, scriere sau ambele. Pentru a crea un fișier nou, este utilizat parametrul *O_CREAT*.

Descriptorul returnat al fișierului poate fi folosit pentru citire sau scriere. După, fișierul poate fi închis prin *close*, iar descriptorul va fi disponibil pentru reutilizare la o deschidere ulterioară.

Cele mai utilizate apeluri sunt, fără îndoială, *read* și *wrtite*. Am discutat despre *read* mai devreme, *write* are aceiași parametri.

Deși majoritatea programelor citesc și scriu fișierele în mod secvențial, pentru unele aplicații programele trebuie să poată accesa oricare parte a unui fișier prin acces direct. Asociat cu fiecare fișier este un indicator (un pointer) care indică poziția curentă din fișier. În cazul unui acces secvențial, acesta indică următorul octet care trebuie citit/scris. Apelul *lseek* schimbă valoarea indicatorlui de poziție, astfel încât apelurile ulterioare pentru citire sau scriere pot începe oriunde în fișier.

Apelul *lseek* are trei parametri: primul este descriptorul fișierului, al doilea este o poziție în fișierului, iar al treilea spune dacă poziția în fișier este relativă la începutul fișierului, la poziția curentă sau la sfârșitul fișierului. Valoarea returnată de *lseek* este poziția absolută în fișier (în octeți) după schimbarea indicatorului de poziție.

Pentru fiecare fișier, UNIX tipul fișierului (fișier obișnuit, fișier special, director și așa mai departe), dimensiunea, timpul ultimei modificări și alte informații. Programele pot cere să vadă aceste informații prin apelul sistemului *stat*. Primul parametru specifică fișierul care trebuie inspectat; cel de-al doilea este un indicator către o structură în care trebuie introduse informațiile. Apelurile *fstat* fac același lucru pentru un fișier deschis.

1.6.3 Apeluri de sistem pentru administrarea cataloagelor

În această secțiune vom analiza unele apeluri de sistem care se referă mai mult la directoare sau la sistemul de fișiere în ansamblu. Primele două apeluri, *mkdir* și *rmdir*, sunt pentru crearea și ;tergerea directoarelor goale. Următorul apel este *link*. Scopul său este de a permite același fișier să apară sub două sau mai multe nume, adesea în directoare diferite. O utilizare tipică este de a permite mai multor membri ai aceleiași echipe de programare să partajeze un fișier comun, pentru fiecare membru fișierul să apară în propriul său director, eventual sub nume diferite. Partajarea unui fișier nu este aceeași cu acordarea unei copii private fiecărui membru

al echipei; a avea un fișier partajat înseamnă că modificările pe care orice membru al echipei le face sunt vizibile instantaneu celorlalți membri - există un singur fișier. Când se lucrează cu copii ale unui fișier, modificările ulterioare aduse unei copii nu le afectează pe celelalte.

Pentru a vedea cum funcționează *link*, luați în considerare situația din fig. 1-21 (a). Există doi utilizatori, *ast* și *jim*, fiecare având propriul său director cu unele fișiere. Dacă *ast* execută un program care conține apelul de sistem

link("*/usr/jim/memo*", "*/usr/ast/note*");

fișierul *memo* din directorul lui jim este acum introdus în directorul ast sub numele *note*. După asta, */usr/jim/memo* și */usr/ast/note* se referă la același fișier. Apropo, unde sunt păstrate directoarele de utilizator (în */usr*, */user*, */home* sau în altă parte) este o decizie luată de administratorul local.

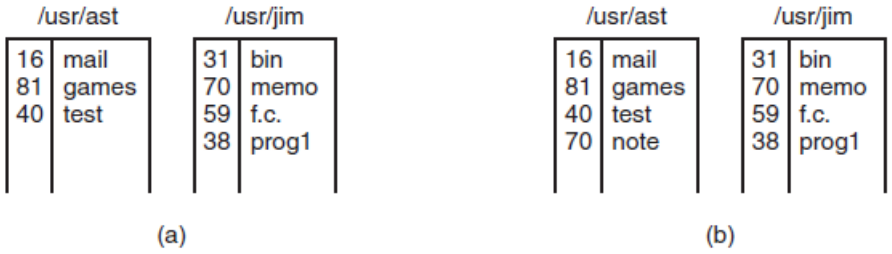

Fig. 1-21. Două directoare până la apelul *link* (a) și după (b)

Înțelegerea modului de funcționare a apelului *link* va clarifica ceea ce acesta face. Fiecare fișier din UNIX are un număr unic, numele său intern *i*, care îl identifică. Acest număr *i* este un index într-un tabel de *i-noduri*, unul per fișier, care spune cine deține fișierul, care sunt blocurile sale de disc și așa mai departe. Un director este pur și simplu un fișier care conține un set de perechi (*i*, nume ASCII). În primele versiuni aleSO UNIX, fiecare intrare de director era 16 octeți - 2 octeți pentru numărul *i* și 14 octeți pentru nume. Acum este necesară o structură mai complicată pentru a susține nume de fișiere lungi, dar conceptual un director este încă un set de perechi (*i*, ASCII). În fig. 1-21, poștă are numărul *i* egal cu 16 și așa mai departe. Ceea ce face *link* este pur și simplu să creeze o intrare de director nou cu un nume (eventual nou), folosind numărul *i* al unui fișier existent. În fig. 1-21 (b), două intrări au același număr *i* (70) și se referă astfel la același fișier. Dacă oricare dintre acestea este eliminat ulterior, folosind apelul de sistem *unlink*, celălalt rămâne. Dacă ambele sunt eliminate, UNIX vede că nu există întrări la fișier (un câmp din nodul *i* ține evidența numărului de intrări de directoare îndreptate către fișier), și fișierul este eliminat de pe disc.

Așa cum am menționat anterior, apelul de sistem *mount* permite îmbinarea a două sisteme de fișiere într-unul. O situație obișnuită este să ai sistemul de fișiere rădăcină, care să conțină versiunile binare (executabile) ale comenzilor comune și alte fișiere folosite în mod intens, pe un hard disk (sub)partiție și fișiere utilizator pe o altă (sub)partiție. Mai departe, utilizatorul poate apoi să insereze un disc USB cu fișiere care să fie citite.

Prin executarea apelului de sistem de *mount*, sistemul de fișiere USB poate fi atașat la sistemul de fișiere rădăcină, așa cum este arătat în fig. 1-22. Un operator tipic *mount* în C este

mount("*/dev/sdb*0", "*/mnt*", 0);

unde primul parametru este numele unui fișier special de bloc pentru unitatea USB 0, cel de-al doilea parametru este locul din arborele unde urmează să fie montat, iar al treilea parametru spune dacă sistemul de fișiere trebuie montat pentru citire și scriere sau doar pentru citire.

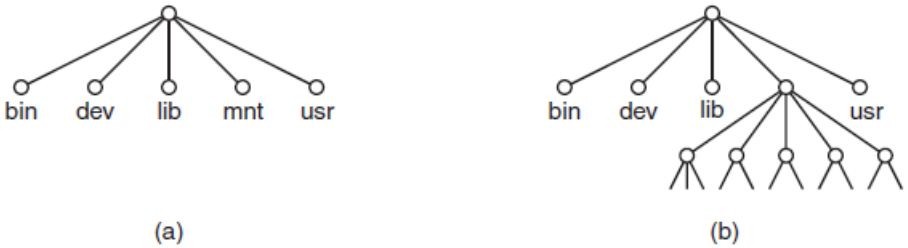

Figura 1-22. (a) Sistem de fișiere înainte de montare. (b) Sistem de fișiere după montare.

După apelul *mount*, un fișier de pe unitatea 0 poate fi accesat doar utilizând calea sa din directorul rădăcină sau din directorul curent, fără a ține cont de unitatea inițială. Un al doilea, al treilea sau al patrulea dispozitv poate fi, de asemenea, montat oriunde în arbore. Apelul *mount* face posibilă integrarea suportului amovibil într-o singură ierarhie de fișiere integrate, fără a fi nevoie să vă faceți griji cu privire la dispozitivul pe care este activat un fișier. Deși acest exemplu se referea la CD-ROM-uri, porțiuni de hard disk-uri (adesea numite partiții sau dispozitive secundare) pot fi, de asemenea, montate în acest fel, precum și hard disk-uri externe și stick-uri USB. Când un sistem de fișiere nu mai este necesar, acesta poate fi demontat cu apelul de sistem *umount*.

1.6.4 Alte apeluri de sistem

Există o varietate de alte apeluri de sistem. Vom analiza doar patru dintre ele aici. Apelul *chdir* modifică directorul de lucru curent. După apelul

chdir(„*/usr/ast/test*”);

o deschidere a fișierului *xyz* va deschide */usr/ast/test/xyz*. Conceptul de director de lucru (director curent) ne eliberează de nevoia de a tasta tot timpul nume (lungi) absolute de cale.

În UNIX, pentru fiecare fișier există o modalitate folosită pentru protecție. În acest scop sunt folosiți biții de citire-scriere-execuție pentru proprietar, grup și alții. Apelul sistemului *chmod* face posibilă schimbarea modului de accesare a unui fișier. De exemplu, pentru a permite tuturor, cu excepția proprietarului, accesul în regim read-only la un fișier introducem comanda

chmod(„*fișier*”, 0644);

Apelul de sistem *kill* este modul în care utilizatorii și procesele utilizatorului trimit semnale. Dacă un proces este pregătit pentru a intercepta un anumit semnal, atunci când acesta este generat, o procedură de tratare (un handler) va fi lansată în execuție. Dacă procesul nu este pregătit pentru a gestiona semnalul, atunci sosirea lui “ucide” procesul (de unde și numele apelului).

POSIX definește o serie de proceduri pentru a duce evidența timpului. De exemplu, *time* întoarce ora curentă în secunde, cu valoare 0 corespunzătoare datei de 1 ianuarie 1970 la miezul nopții (la începutul zile). Pe computerele care folosesc cuvinte pe 32 de biți, valoarea maximă a timpului este de 2^{32} - 1 secunde. Această valoare este un pic mai mare de 136 ani.

1.6.5 Interfața pentru programarea aplicațiilor Windows Win32 API

Până acum ne-am concentrat în principal pe UNIX. Acum este timpul să privim un pic spre Windows. Windows și UNIX diferă în mod fundamental în modelele de programare respective. Un program UNIX constă dintr-un cod care face ceva sau altul, făcând apeluri de sistem pentru a efectua anumite servicii. În schimb, un program Windows este condus în mod normal de evenimente. Programul principal așteaptă să se întâmple un eveniment, apoi apelează la o procedură (handler-ul) care să-l gestioneze. Evenimentele tipice sunt apăsarea tastelor sau a

butoanelor de mouse, deplasarea mouse-ului sau introducerea unei unități USB, etc. Handlerele de evenimente sunt apelate să proceseze evenimentul, să actualizeze ecranul și să actualizeze starea internă a programului. În total, acest lucru duce la un stil de programare oarecum diferit decât în cazul SO UNIX, dar, întrucât această carte se concentrează asupra funcției și structurii sistemului de operare, aceste modele de programare diferite nu ne vor preocupa prea mult.

Desigur, Windows are și apeluri de sistem. Cu UNIX, există aproape o relație unu la unu între apelurile de sistem (de exemplu, *read*) și procedurile de bibliotecă (de exemplu, **`read`**) utilizate pentru a executa apelurile de sistem. Cu alte cuvinte, pentru fiecare apel de sistem, există aproximativ o procedură de bibliotecă care este chemată să îl execute, așa cum este indicat în fig. 1-17. Mai mult, POSIX include doar aproximativ 100 de apeluri de procedură.

Cu Windows, situația este radical diferită. Pentru început, apelurile din bibliotecă și apelurile reale ale sistemului sunt foarte decuplate. Microsoft a definit un set de proceduri numite API Win32 (Application Programming Interface) pe care programatorii trebuie să le utilizeze pentru a obține serviciile sistemului de operare. Această interfață este (parțial) acceptată pe toate versiunile Windows începând cu Windows 95. Prin decuplarea interfeței API de la apelurile reale ale sistemului, Microsoft își păstrează posibilitatea de a schimba apelurile reale ale sistemului la timp (chiar și de la lansare la lansare) fără a invalida programele existente. Ceea ce constituie de fapt Win32 este, în principiu, ușor ambiguu, deoarece versiunile recente ale Windows au numeroase apeluri noi care nu au fost disponibile anterior. În această secțiune, Win32 înseamnă interfața acceptată de toate versiunile de Windows. Win32 oferă compatibilitate între versiunile Windows.

Numărul de apeluri API Win32 este extrem de mare, de ordinul miilor. Mai mult, în timp ce multe dintre apelurile API Win21 invocă apeluri de sistem, un număr substanțial este efectuat în întregime în spațiul utilizatorului. În consecință, cu Windows este imposibil de văzut ce este un apel de sistem (adică, efectuat de kernel) și ce este pur și simplu un apel de bibliotecă în spațiul utilizatorului. De fapt, ceea ce este un apel de sistem într-o versiune de Windows se poate face în spațiul utilizatorului într-o versiune diferită, și invers. Cand discutăm apelurile sistemului Windows din această carte, vom folosi procedurile Win32 (acolo unde este cazul), deoarece Microsoft garantează că acestea vor fi stabile în timp. Dar merită să ne amintim că nu toate sunt adevărate apeluri de sistem (adică Traps (capcane) către kernel).

API-ul Win32 are un număr mare de apeluri pentru gestionarea ferestrelor, figurilor geometrice, textului, fonturilor, barelor de defilare, casetelor de dialog, meniurilor și a altor caracteristici ale GUI. În măsura în care subsistemul grafic rulează în kernel (adevărat pe unele versiuni de Windows, dar nu pe toate), acestea sunt apeluri de sistem; altfel sunt doar apeluri la bibliotecă. Ar trebui să discutăm sau nu aceste apeluri în această carte? Deoarece nu au legătură cu funcția unui sistem de operare, am decis să nu o facem, chiar dacă acestea pot fi efectuate de kernel. Cititorii interesați de API-ul Win32 ar trebui să consulte una dintre numeroasele cărți pe această temă (de exemplu, Hart, 1997; Rector și nou-venit, 1997; și Simon, 1997).

Dacă introducerea tuturor apelurilor API Win32 nu se pune în discuție, ne vom limita la acele apeluri care corespund aproximativ funcționalității apelurilor UNIX enumerate în fig. 1-18. Acestea sunt enumerate în fig. 1-23. Să trecem acum pe scurt pe lista din fig. 1-23. *CreateProcess* creează un nou proces. Face lucrul combinat al *fork* și *execve* din UNIX. Are mulți parametri care specifică proprietățile procesului nou creat. Windows nu are o ierarhie de procese așa cum este în UNIX, astfel nu există niciun concept de proces părinte și proces copil. După crearea unui proces, descendentul și creatorul sunt egali. *WaitForSingleObject* este folosit pentru a aștepta un eveniment. Multe evenimente posibile pot fi așteptate. Dacă

parametrul specifică un proces, atunci apelantul așteaptă ieșirea procesului specificat, care se realizează folosind *ExitProcess*.

Următoarele șase apeluri operează cu fișierele și sunt asemănătoare funcțional cu omologii lor UNIX, deși diferă în parametri și detalii. Totuși, fișierele pot fi deschise, închise, citite și scrise aproape ca în UNIX. Apelurile *SetFilePointer* și *GetFileAttributesEx* setează poziția fișierului sau obțin atribute ale fișierului.

Windows are directoare care sunt create cu apelând *CreateDirectory* și distruse apelând *RemoveDirector* din Win32 API. Există, de asemenea, o noțiune de director curent, setată cu ajutorul lui *SetCurrentDirectory*. Ora curentă a zilei este obținută folosind *GetLocalTime*.

Interfața Win32 nu are legături către fișiere, sisteme de fișiere montate, securitate sau semnale, astfel încât apelurile corespunzătoare celor UNIX nu există. Desigur, Win32 are un număr foarte mare de alte apeluri pe care UNIX nu le are, în special pentru gestionarea GUI. Ultimile versiuni Windows Vista au un sistem puternic de securitate, acceptă legături de fișiere și multe alte funcții și apeluri de sistem.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount, so no umount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Fig. 1-23. Apelurile Win32 API, care corespund apelurilor UNIX din fi. 1-18.

O ultimă observație despre Win32 merită a fi făcută. Win32 nu este o interfață teribil de uniformă sau consistentă. Principalul vinovat a fost nevoia de a fi compatibil cu interfața anterioară pe 16 biți folosită în Windows 3.x.

1.7 STRUCTURA SISTEMELOR DE OPERARE

După ce am văzut cum arată sistemele de operare la exterior (adică la nivel de interfață a programatorului), este timpul să aruncăm o privire în interior. În secțiunile următoare, vom examina șase structuri diferite care au fost încercate, pentru a ne face o idee despre spectrul

posibilităților. Acestea nu sunt în niciun caz exhaustive, dar oferă o idee a unor modele care au fost încercate în practică. Cele șase modele despre care vom discuta aici sunt sistemele monolitice, sistemele stratificate, microkernelele, sistemele client-server, mașinile virtuale și exokernelul.

1.7.1 Sisteme monolitice

Este de departe cea mai comună organizare. În demersul monolitic întregul sistem de operare rulează ca un singur program în modul kernel. Sistemul de operare este scris ca o colecție de proceduri, legate între ele într-un singur program binar executabil mare. Când se folosește această tehnică, fiecare procedură din sistem este liberă să apeleze oricare alta, dacă aceasta din urmă furnizează unele calcule utile de care are nevoie prima. Posibilitatea de a apela orice procedură este foarte eficientă, dar a avea mii de proceduri care se pot apela reciproc fără restricții poate duce la un sistem complicat și greu de înțeles. De asemenea, o cădere în oricare dintre aceste proceduri va conduce la căderea întregului sistem de operare.

Pentru a construi modulul obiect al sistemului de operare atunci când este utilizată această abordare, mai întâi trebuie compilate toate procedurile individuale (sau fișierele care conțin procedurile) care apoi vor fi legate într-un singur fișier executabil utilizând link-erul de sistem. În termeni de ascundere a informațiilor, ascunderea nu există - fiecare procedură este vizibilă pentru orice altă procedură (spre deosebire de o structură cu module sau pachete, în care o mare parte din informație este ascunsă în interiorul modulelor și numai punctele de intrare desemnate oficial pot fi apelate din afara modulului).

Chiar și în sistemele monolitice este posibil să existe o anumită structură. Serviciile (apeluri de sistem) furnizate de sistemul de operare sunt solicitate prin plasarea parametrilor într-un loc bine definit (de exemplu, pe stivă) cu executarea unei instrucțiuni de interceptare. Această instrucțiune trece execuția de la modul utilizator la modul kernel și transferă controlul la sistemul de operare, ca pasul 6 din fig. 1-17. Apoi, sistemul de operare preia parametrii și determină ce apel de sistem trebuie efectuat și lansează procedura care efectuează apelul de sistem *k* (pasul 7 din fig. 1-17).

Această organizație sugerează următoarea structură pentru sistemul de operare:

1. Un program principal care invocă procedura de service solicitată.
2. Un set de proceduri de service care efectuează apelurile de sistem.
3. Un set de proceduri utilitare care ajută procedurile de servicii.

În acest model, pentru fiecare apel de sistem există o singură procedură de serviciu care are grijă de ea și o execută. Procedurile de utilitate fac lucrurile necesare mai multor proceduri de servicii, cum ar fi preluarea datelor din programele utilizatorului. Această divizare a procedurilor în trei straturi este prezentată în fig. 1-24.

În plus față de sistemul de operare de bază care este încărcat la pornirea computerului, multe sisteme de operare acceptă extensii care pot fi încărcate, cum ar fi driverele de dispozitive I/E și sistemele de fișiere. Aceste componente sunt încărcate la cerere. În UNIX se numesc

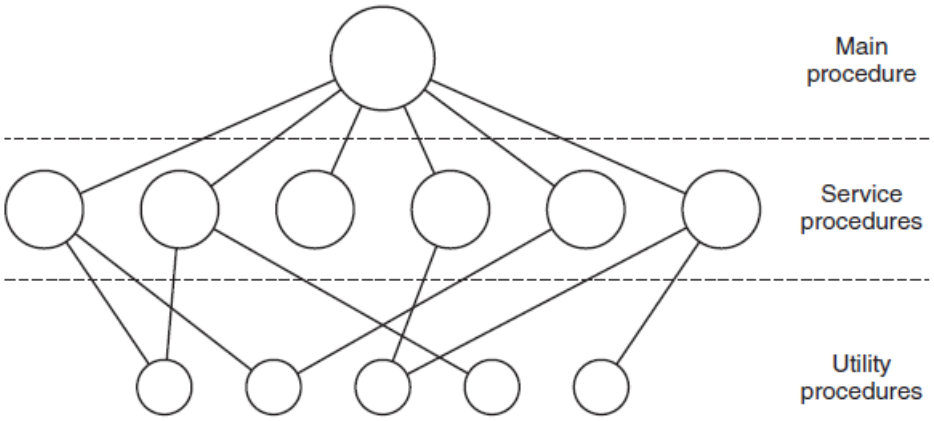

Fig. 1-24. Structura unui sistem de operare monolitic

biblioteci partajate. În Windows se numesc DLL (Dynamic-Link Libraries), au extensia de fișier .dll și în directorul C:\Windows\system32 din Windows pot fi găsite peste 1000.

1.7.2 Sisteme structurate pe nivele

O generalizare a abordării din fig. 1-24 este de a organiza sistemul de operare ca o ierarhie de straturi, fiecare construit pe cel de sub el. Primul sistem construit în acest fel a fost sistemul THE construit la Technische Hogeschool Eindhoven din Olanda de E. W. Dijkstra (1968) și elevii săi, care era un sistem simplu, pentru un computer olandez, Electrologica X8, cu 32K de cuvinte pe 27 de biți (biții erau scumpi pe atunci). Sistemul avea șase nivele, așa cum este aratat în fig. 1-25. Stratul 0 era responsabil de alocarea procesorului - comutarea între procese atunci când aveau loc întreruperi sau expira cuanta de alocare. Peste nivelul 0 erau procesele secvențiale fiecare în spațiul propriu de memorie, mai multe procese rulau pe un singur procesor. Cu alte cuvinte, nevelul 0 forma baza pentru multiprogramarea procesorului.

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Fig. 1-25. Structura sistemului de operare THE

Nivelul 1 era responsabil de managementul memoriei. Aici era alocat spațiu pentru procesele din memoria principală și pentru un tambur de 512K cuvinte folosit pentru păstrarea unor părți ale proceselor (pagini) pentru care nu exista suficient spațiu în memoria principală. Mai sus de nivelul 1, procesele nu trebuiau să se îngrijoreze dacă sunt în memorie sau în această memorie tambur; software-ul stratului 1 asigura aducerea paginilor în memoria principală la necesitate sau eliminarea lor atunci când nu era necesare.

Nivelul 2 era nivelul care gestiona comunicărilor între procese și consola operatorului. Acest nivel punea la dispoziția fiecărui proces propria sa consolă operator. Nivelul 3 se ocupa de gestiunea dispozitivelor I/O și de buferizarea fluxurilor de date către și de la acestea. Datorită nivelului 3, fiecare proces trata dispozitive de I/O abstracte cu proprietăți frumoase, în loc de dispozitive reale cu multe particularități.

Nivelul 4 era al programelor de utilizator. Utilizatorii nu aveau grijă de gestionarea proceselor, memoriei, consolei sau I/O. Procesul operatorului de sistem era localizat în nivelul 5.

O nouă generalizare a conceptului de structură ultinivel a fost prezentă în sistemul MULTICS. În loc de nivele, MULTICS avea o serie de inele concentrice, cele interioare fiind mai privilegiate decât cele exterioare (ceea ce este efectiv același lucru). Când o procedură dintr-un inel exterior dorea să apeleze la o procedură într-un inel interior, ea trebuia să facă echivalentul unui apel de sistem, adică o instrucțiune TRAP care verifica cu atenție validitatea parametrilor transmiși înainte ca apelul să poată continua. Deși întregul sistem de operare făcea parte din spațiul de adrese al fiecărui proces de utilizator MULTICS, hardware-ul desemna procedurile individual (segmente de memorie, de fapt) ca protejate împotriva citirii, scrierii sau executării.

1.7.3 Modelul microkernel

Prin abordarea multinivel designerii pot stabili unde să fie plasată granița user-kernel. În mod tradițional, toate straturile intrau în kernel, dar acest lucru nu este necesar. De fapt, ar fi fost mult mai binevenit să se pună cât mai puține obligațiuni nivelului kernel, deoarece bug-urile din kernel pot conduce la căderea întregului sistem. În schimb, procesele utilizatorilor pot fi configurate astfel încât să aibă mai puțină putere pentru ca erorile de aici să nu fie fatale.

Diversi cercetători au studiat în mod repetat statisticile referitoare la numărul de buguri la 1000 de linii de cod (de exemplu, Basilli și Perricone, 1984; și Ostrand și Weyuker, 2002). Densitatea erorilor depinde de dimensiunea modulului, vârsta modulului și multe altele, dar o valoare estimativă pentru sistemele industriale mari este cuprinsă între două și zece buguri la o mie de linii de cod. Aceasta înseamnă că un sistem de operare monolitic cu cinci milioane de linii de cod poate conține între 10.000 și 50.000 de erori de kernel. Nu toate acestea sunt fatale, desigur, deoarece unele bug-uri pot fi lucruri precum emiterea unui mesaj de eroare incorect într-o situație care apare rar. Cu toate acestea, sistemele de operare conțin suficiente bug-uri încât producătorii de computere să pună butoane de resetare, ceva ce producătorii de televizoare sau automobile nu fac, în ciuda cantității mari de software din acestea.

Ideea de bază din spatele proiectării microkernelului este de a obține o fiabilitate ridicată prin împărțirea sistemului de operare în module mici, bine definite, doar unul dintre ele - microkernelul - rulează în modul kernel, iar restul rulează ca procese de utilizator obișnuite. În special, rulând fiecare driver de dispozitiv sau sistem de fișiere ca proces de utilizator separat, o eroare într-unul dintre acestea poate bloca acea componentă, dar nu poate duce la căderea întregului sistem. Astfel, o eroare a driverului audio va face ca sunetul să fie de calitate joasă sau să se oprească, dar nu va bloca întregul calculator.

Pe parcursul anilor au fost realizate și implementate mai multe sisteme cu micronucleu (Haertig și colab., 1997; Heiser și colab., 2006; Herder și colab., 2006; Hildebrand, 1992; Kirsch și colab., 2005; Liedtke, 1993, 1995, 1996; Pike și colab., 1992; și Zuberi și colab., 1999). Cu excepția OS X, care se bazează pe microkernel-ul Mach (Accetta și colab., 1986), sistemele de operare obișnuite pentru desktop nu folosesc micro-nuclee. Cu toate acestea, ele sunt dominante în aplicații de timp real, industrial, avionic și militar, critice pentru misiune și au cerințe de fiabilitate foarte ridicate. Câteva dintre cele mai cunoscute microkerneluri includ Integrity, K42, L4, PikeOS, QNX, Symbian și MINIX 3. Vom oferi în continuare o scurtă privire de ansamblu asupra MINIX 3, care a dus ideea modularității la limită, împărțind cea mai mare parte a sistemului de operare în mai multe procese independente în modul utilizator. MINIX 3 este un sistem open source conform POSIX, disponibil gratuit pe www.minix3.org (Giuffrida și colab., 2012; Giuffrida și colab., 2013; Herder și colab., 2006; Herder și colab., 2009; și Hruby et al., 2013).

Micronucleul MINIX 3 reprezintă aproximativ 12.000 de linii de cod C și aproximativ 1400 de linii cod de asamblare pentru funcții de nivel foarte scăzut, cum ar fi ointerceptarea întreruperilor și procesele de comutare. Codul C gestionează și planifică procesele, gestionează comunicarea intre procese (prin mesaje) și oferă un set de aproximativ 40 de apeluri kernel pentru a permite restului sistemului de operare să își facă treaba. Aceste apeluri îndeplinesc funcții precum legarea handlerului de o întrerupere, mutarea datelor între spațiile de adrese și instalarea hărților de memorie pentru procesele noi. Structura procesului MINIX 3 este prezentată în fig. 1-26 în care apelul de kernel este etichetat cu *Sys*. Driverul dispozitivului pentru ceas este, de asemenea, în kernel, deoarece planificatorul interacționează strâns cu acesta. Celelalte drivere de dispozitiv rulează ca procese de utilizator separate.

În afara nucleului, sistemul este structurat pe trei nivele de procese, toate rulând în modul utilizator. Cel mai jos nivel conține driverele de dispozitiv. Întrucât rulează în modul utilizator, nu au acces fizic la spațiul portului I/O și nu pot emite comenzi I/O direct. În schimb, pentru a programa un dispozitiv de I/O, driverul construiește o structură care spune ce valori trebuie să scrie la care porturi I/O și face un apel al kernelului care îi spune kernelului să execute scrierea. Această abordare înseamnă că nucleul poate verifica pentru a vedea dacă driverul scrie (sau citește) cu I/O pe care este autorizat să o folosească. În consecință (și spre deosebire de un design monolitic), un driver audio buggy nu poate scrie accidental pe disc.

Deasupra driverelor este un alt nivel de mod utilizator care conține servere, care realizează cea mai mare parte a activității sistemului de operare. Unul sau mai multe servere de fișiere gestionează sistemul (sistemele) de fișiere, managerul de procese creează, distruge și gestionează procesele etc. Programele utilizator obțin servicii ale sistemului de operare prin trimiterea de mesaje scurte către servere. De exemplu, un proces care trebuie să facă o citire trimite un mesaj unuia dintre serverele de fișiere spunându-i ce să citească.

Un server interesant este serverul de reîncarnare, a cărui sarcină este să verifice dacă celelalte servere și drivere funcționează corect. În cazul în care se detectează unul defect, acesta se înlocuiește automat fără nicio intervenție a utilizatorului. În acest fel, sistemul se autotratează, operație care sporește semnificativ fiabilitatea.

Sistemul are multe restricții care limitează posibilitățile fiecărui proces. Așa cum am menționat, driverele pot atinge doar porturi de I/O autorizate, accesul la apelurile kernelului este controlat pe bază de proces. Procesele pot acorda, de asemenea, permisiunea limitată pentru alte procese pentru ca nucleul să aibă acces la spațiile de adrese. Ca exemplu, un sistem de fișiere poate acorda permisiunea pentru driverul de disc să permită nucleului să pună un bloc de disc nou citit la o adresă specifică din spațiul de adrese al sistemului de fișiere. Suma totală a tuturor acestor restricții este aceea că fiecare driver și server are exact puterea de a-și face munca și nimic mai mult, limitând astfel foarte mult daunele pe care le poate face o componentă buggy.

Un nucleu poate fi minimizat dacă va fi responsabil doar de mecanisme, nu și de politici. De exemplu, dacă utilizăm priorități pentru procese poate fi creat un algoritm de planificare relativ simplu. Nucleul va rula procesul cu prioritate maximă. Mecanismul - în kernel - este să caute procesul cu prioritate cea mai înaltă și să îl execute. Politica, adică cum să fie alocate prioritățile proceselor, - poate fi realizată prin procese executate în modul utilizator. În acest fel, politica și mecanismul pot fi decuplate și nucleul poate fi redus.

1.7.4 Modelul client-server

O ușoară variație a ideii de microkernel este de a separa procesele în două clase: clasa proceselor server, fiecare oferind un serviciu și clasa proceselor client care utilizează aceste servicii. Acest model este cunoscut sub numele de modelul client-server. Adesea, cel mai jos

nivel este un microkernel, dar acest lucru nu este obligator. Esența constă în prezența proceselor client și a proceselor server.

Comunicarea între clienți și servere se face adesea prin transmiterea mesajelor. Pentru a obține un serviciu, un proces client construiește un mesaj care spune ce dorește și îl trimite serverului corespunzător. Serverul tratează cererea și trimite înapoi răspunsul. Dacă procesele client și server sunt executate pe aceeași mașină, sunt posibile anumite optimizări, dar conceptual, rămânem pe aceeași lungume de undă.

O generalizare evidentă a acestei idei este ca clienții și serverele să fie rulate pe diferite calculatoare, conectate printr-o rețea locală sau globală, așa cum este prezentat în fig. 1-27. Deoarece clienții comunică cu serverele prin trimiterea de mesaje, clienții nu sunt obligați să știe dacă mesajele sunt gestionate local pe propriile mașini sau dacă sunt trimise prin intermediul unei rețele către servere de pe o mașină de la distanță. În ceea ce privește clientul, același lucru se întâmplă în ambele cazuri: cererile sunt trimise și răspunsurile revin. Astfel, modelul client-server este o abstractizare care poate fi folosită pentru o singură mașină sau pentru o rețea de mașini.

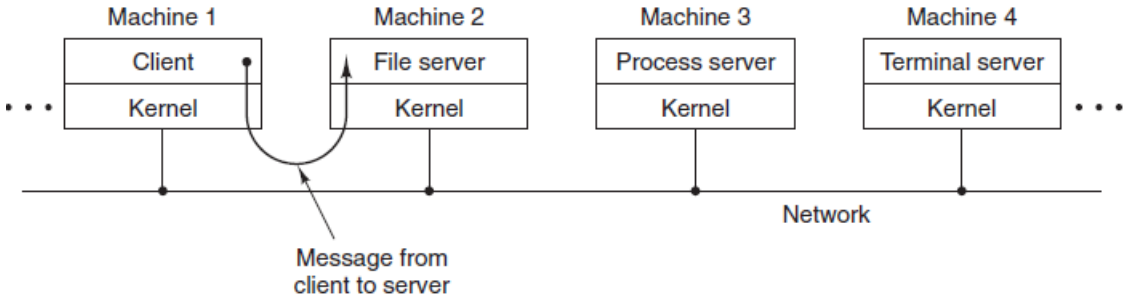

Fig. 1-27. Modelul client-server pentru o rețea de calculatoare

Este modul de lucru modern în care tot mai mulți utilizatori, folosind calculatoarele lor de acasă - clienți, solicită serviciile unor mașini mari care funcționează în altă parte - servere. De fapt, o mare parte din Web funcționează astfel. Un computer trimite o solicitare pentru o pagină Web către server și pagina web revine. Este utilizarea obișnuită a modelului client-server într-o rețea.

1.7.5 Masini virtuale

Versiunile inițiale ale sistemului de operare OS/360 erau pentru prelucrarea pe loturi. Cu toate acestea, mulți de utilizatori de OS/360 doreau să poată lucra în mod interactiv la un terminal, astfel încât diverse grupuri, atât din interiorul cât și din afara IBM, au decis să scrie sisteme cu timp partajat în acest scop. Sistemul oficial IBM cu timesharing, TSS/360, a cam întârziat, iar atunci când în sfârșit a fost prezentat, era de mare și încet încât puțini au fost cei care l-au acceptat. Într-un final a fost abandonat, chiar dacă dezvoltarea sa a constituit aproximativ 50 de milioane de dolari (Graham, 1970). Dar un grup de dezvoltatori de la IBM Scientific Center din Cambridge, Massachusetts, a propus un sistem radical diferit, acceptat de IBM. Un descendent al acestuia, numit z/VM, este acum utilizat pe scară largă pe mainframe-urile actuale IBM zSeries, mainframe-uri foarte utilizate în centrele de date corporative mari, de exemplu, ca servere de e-commerce care gestionează sute sau mii de tranzacții pe secundă și utilizează baze de date cu dimensiuni de ordinul miilor de terabaiți.

VM/270

Acest sistem, denumit inițial CP/CMS și mai apoi redenumit VM/370 (Seawright și MacKinnon, 1979), s-a bazat pe o observație isteață: un sistem partajarea timpului oferă (1)

multiprogramare și (2) o mașină extinsă cu o interfață mai convenabilă decât hardware-ul gol. Esența VM/370 este separarea completă a acestor două funcții.

Partea centrală a sistemului, cunoscută sub numele de *monitorul mașinii virtuale*, rulează pe hardware-ul gol și realizează multiprogramarea, oferind nu una, ci mai multe mașini virtuale pentru următorul nivel (fig. fig. 1-28). Cu toate acestea, spre deosebire de toate celelalte sisteme de operare, aceste mașini virtuale nu sunt mașini extinse, cu sisteme de fișiere și alte funcții frumoase. În schimb, acestea sunt copii exacte ale hardware-ului gol, inclusiv modul kernel/utilizator, I/O, întreruperi și orice altceva ce are mașina reală.

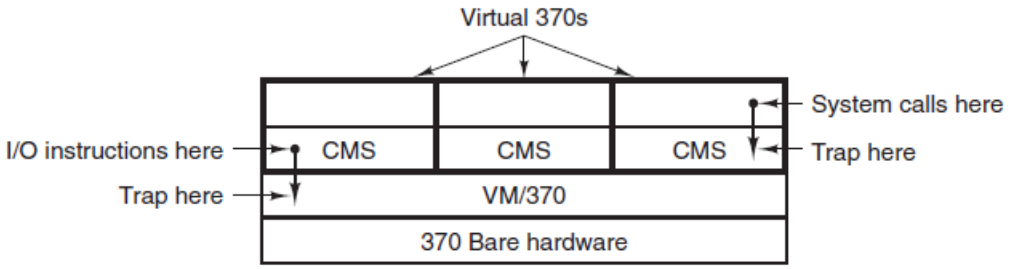

Fig. 1-28. Structura VM/370 cu CMS

Deoarece fiecare mașină virtuală este identică cu hardware-ul adevărat, pe fiecare poate fi instalat orice sistem de operare care va rula direct pe hardware-ul gol. Unii utiizatori IBM VM/370 rulau OS/360, sisteme cu procesare pe loturi sau pentru procesarea tranzacțiilor, în timp ce alții au rulau un sistem interactiv cu un singur utilizator, numit CMS (Conversational Monitor System). Acesta din urmă era foarte popular în rândul programatorilor.

Când un program CMS executa un apel de sistem, apelul era interceptat de sistemul de operare în propria sa mașină virtuală (trap), nu de VM/370. CMS emula instrucțiunile de I/O hardware normale pentru citirea discului său virtual sau orice era necesar pentru a executa apelul. Aceste instrucțiuni de I/O eau apoi preluate de VM/370, care le executa ca parte a simulării hardware-ului real. Prin separarea completă a funcțiilor (multiprogramare și furnizarea mașinii extinseI, fiecare dintre componente deveneau mai simple, mai flexibile și mult mai ușor de întreținut.

În realizarea sa modernă, z/VM este utilizat pentru a rula mai multe sisteme de operare complete, mult mai performante decât sistemele cu un singur utilizator cum ar fi CMS. De exemplu, sistemul zSeries este capabil să ruleze una sau mai multe mașini virtuale Linux împreună cu sistemele de operare tradiționale IBM.

Mașini virtuale redescoperite

Deși IBM a implementat mașina virtuală cu aproape 5 decenii în urmă, iar câteva alte companii, inclusiv Oracle și Hewlett-Packard, au adăugat recent suport pentru mașini virtuale pe serverele lor de înaltă performanță, ideea virtualizării a fost în mare parte ignorată în lumea PC-urilor. Dar în ultimii ani, o combinație de necesități, tehnologii și software noi s-au combinat și au creat un subiect fierbinte.

Mai întâi despre nevoi. Multe companii și-au rulat în mod tradițional serverele de poștă, serverele Web, serverele FTP și alte servere pe computere separate, uneori cu sisteme de operare diferite. Ei văd virtualizarea ca o modalitate de a le rula pe toate pe aceeași mașină fără ca o cădere a unui server să codnucă la căderea celorlalte servere.

Virtualizarea este de asemenea populară în hosting-ul web. Fără virtualizare, clienții care au nevoie de hosting web sunt nevoiți să aleagă între hosting partajat (care le oferă doar un cont

de autentificare pe un server Web, dar fără control asupra software-ului serverului) și hosting dedicat (ceea ce le oferă propria mașină, care este foarte flexibilă, dar acest mod nu este recomandat pentru site-urile mici până la mijlocii). Atunci când o companie de hosting web oferă mașini virtuale de închiriat, o singură mașină fizică poate rula mai multe mașini virtuale, fiecare dintre ele simulând o mașină separată. Clienții pot folosi orice sistem de operare sau software aplicativ, achitând doar o parte din costul unui server dedicat.

O altă utilizare a virtualizării este pentru utilizatorii finali care doresc să poată rula două sau mai multe sisteme de operare în același timp, deoarece unele dintre pachetele de aplicații preferate rulează pe un sistem, iar altele - pe celălalt. Această situație este ilustrată în fig. 1-29 (a), unde termenul “monitor de mașină virtuală” a fost redenumit în *hipervizor de tip 1*.

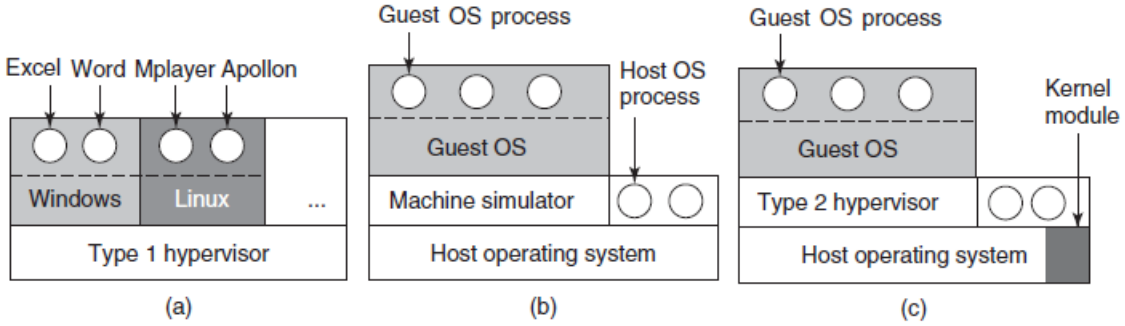

Fig. 1-29. (a) Hipervizor de tip 1. (b) Hipervizor pur de tip 2. (c) Hipervizor practic de tip 2

Deși astăzi nimeni nu contestă atractivitatea mașinilor virtuale, la momentul realizării ideii problema era. Pentru a rula software-ul mașinii virtuale pe un computer, procesorul său trebuie să fie virtualizabil (Popek și Goldberg, 1974) - aici este problema. Când un sistem de operare care rulează pe o mașină virtuală (în modul utilizator) execută o instrucțiune privilegiată, cum ar fi modificarea PSW sau efectuarea I/O, este esențial ca trap-ul hardware să fie monitorizat pentru mașina virtuală, astfel încât instrucțiunea să poată fi emulată în software. Pe unele procesoare - în special Pentium, predecesorii și clonele sale - încercările de a executa instrucțiuni privilegiate în modul utilizator sunt doar ignorate. Această proprietate a făcut imposibil existența de mașini virtuale pe acest hardware, ceea ce explică lipsa de interes pentru lumea x86. Simulatoarele pentru Pentium, cum ar fi Bochs, care rulau pe Pentium aveau o pierdere de performanță de la 10 până la o sută de ori și nu au fost utili pentru munci serioase.

Această situație s-a schimbat ca urmare a mai multor proiecte de cercetare academică din anii 1990 și primii ani ai acestui mileniu, în special Disco la Stanford (Bugnion și colab., 1997) și Xen la Universitatea Cambridge (Barham și colab., 2003). Aceste lucrări de cercetare au dus la mai multe produse comerciale (de exemplu, VMware Workstation și Xen) și la o renaștere a interesului pentru mașinile virtuale. Pe lângă VMware și Xen, hipervizorii populari includ astăzi KVM (pentru nucleul Linux), VirtualBox (de Oracle) și Hyper-V (de Microsoft).

Unele dintre aceste proiecte de cercetare timpurie au sporit performanța comparativ cu Bochs prin compilarea blocurilor de cod „din zbor”, stocarea lor într-o memorie cache internă și apoi reutilizarea lor (dacă nu au fost modificate). Acest lucru a îmbunătățit considerabil performanța și a dus la ceea ce vom numi simulatoare de mașini (fig. 1-29 (b)). Cu toate acestea, deși această tehnică, cunoscută sub numele de translatare binară, a ajutat la îmbunătățirea situației, sistemele rezultate, deși sunt suficient de bune pentru a publica lucrări la conferințe, nu sunt încă destul de rapide pentru a fi utilizate în medii comerciale unde performanța contează mult.

Următorul pas în îmbunătățirea performanței a fost adăugarea unui modul de kernel (fig. 1-29 (c)). În practică, toți hipervizorii comerciali, cum ar fi VMware Workstation, folosesc această

strategie hibridă (plus multe alte îmbunătățiri). Sunt numiți hipervizori de tip 2 de toată lumea, așa că vom folosi această denumire în restul acestei cărți, chiar dacă am prefera să le numim hipervizori de tip 1.7 pentru a reflecta faptul că nu sunt în întregime programe executate în mod user. În cap. 7, vom descrie în detaliu cum funcționează VMware Workstation.

Mașina virtuală Java

Un alt domeniu în care sunt utilizate mașini virtuale, dar într-un mod oarecum diferit, este rularea programelor Java. Când Sun Microsystems a inventat limbajul de programare Java, a inventat și o mașină virtuală (adică, o arhitectură computerizată) numită JVM (Java Virtual Machine). Compilatorul Java produce cod pentru JVM, care este de obicei executat de un interpretor software JVM. Avantajul acestei abordări constă în faptul că codul JVM poate fi expediat pe internet către orice computer care are un interpretor JVM și poate rula acolo. Dacă compilatorul ar fi produs programe binare SPARC sau x86, de exemplu, acestea nu ar fi putut fi expediate și rulate la fel de ușor. (Desigur, Sun ar fi putut propune un compilator care să producă cod binar SPARC și apoi să distribuie un interpretor SPARC, dar JVM este o arhitectură mult mai simplă de interpretat.) Un alt avantaj al utilizării JVM este că dacă interpretorul este implementat corect, ceea ce nu este atât de simplu, programele JVM banale, pot fi verificate pentru siguranță și apoi executate într-un mediu protejat, astfel încât acestea să nu poată fura date sau să facă pagube.

1.7.6 Despre Exokernel

În locul clonării unei mașini, așa cum se face în cazul mașinilor virtuale, există o altă strategie numită partajare, prin care fiecărui utilizator se pune la dispoziție un subset de resurse. Astfel, o mașină virtuală poate primi blocuri de disc de la 0 până la 1023, următoarea ar putea primi blocuri 1024 - 2047 și așa mai departe.

În nivelul de jos, care rulează în modul kernel, se află un program numit exokernel (Engler și colab., 1995). Sarcina sa este să aloce resurse mașinilor virtuale și apoi să verifice încercările de a le folosi pentru a se asigura că nici o mașină nu încearcă să utilizeze resursele altcuiva. Fiecare mașină virtuală la nivel de utilizator poate rula propriul sistem de operare, ca pe VM/370 și Pentium virtual 8086, cu excepția faptului că fiecare este restricționată să utilizeze doar resursele solicitate după ce au fost alocate.

Avantajul schemei exokernel este că economisește un nivel de mapare. În celelalte modele, fiecare mașină virtuală crede că are propriul său disc, cu blocuri care rulează de la 0 la un maxim, astfel încât monitorul mașinii virtuale trebuie să mențină tabele pentru a remapa adresele de disc (și toate celelalte resurse). Cu exokernel-ul, această remapare nu este necesară. Exokernel-ul trebuie doar să țină evidența de la ce mașină virtuală și care resursă i-a fost atribuită. Această metodă mai are avantajul de a separa multiprogramarea (în exokernel) de codul sistemului de operare al utilizatorului (în spațiul utilizatorului).

1.8 DIN LUMEA LIMBAJULUI C

Sistemele de operare constă din programe mari în limbajul C (sau uneori C++), formate din multe module, scrise de mulți programatori. Mediul utilizat pentru dezvoltarea sistemelor de operare este foarte diferit de ceea ce obișnuiesc programatorii simpli (cum ar fi studenții) atunci când scriu mici programe Java. Această secțiune este o încercare de a oferi o scurtă introducere în lumea scrierii unui sistem de operare pentru programatorii Java sau Python.

1.8.1 Despre limbajul C

Acesta nu este un ghid pentru C, ci un scurt rezumat al unora dintre diferențele cheie între C și limbbaje precum Python și în special Java. Java se bazează pe C, deci există multe asemănări între cele două. Python este oarecum diferit, dar are și similitudini. Pentru comoditate, ne vom concentra pe Java. Java, Python și C sunt limbaje imperative cu tipuri de date, variabile și instrucțiuni de control. Tipurile de date primitive din C sunt numerele întregi (inclusiv cele scurte și lungi), caractere și numere în virgulă flotantă. Tipurile de date compuse sunt tablourile, structurile și uniunile. Instrucțiunile de control în C sunt similare cu cele din Java, inclusiv instrucțiunile *if*, *switch*, *for* și *while*. Funcțiile și parametrii sunt aproximativ aceiași în ambele limbaje.

O caracteristică prezentă în C și lipsă în Java și Python pointerii expliciți. Un pointer este o variabilă care indică (conține adresa) unei alte variabile sau a unei structuri de date.

În fragmentul de cod

```
char c1, c2, *p;
c1 = „c”;
p = &c1;
c2 = *p;
```

c1 și c2 sunt declarate ca variabile de tip caracter, iar p ca o variabilă care indică, adică conține adresa unui caracter. Prima atribuire pune codul ASCII al caracterului „c” în variabila c1. A doua atribuie adresa lui c1 variabilei pointer p. A treilea alocă conținutul variabilei punctate de p variabilei c2, astfel încât după executarea acestor declarații, c2 conține codul ASCII al lui „c”. În teorie nu ar trebui să atribuiți adresa unui număr în virgulă flotantă unui indicator de caracter, dar în practică compilatoarle acceptă astfel de atribuiri, deși uneori cu un avertisment. Pointerii sunt o construcție foarte puternică, dar și o mare sursă de erori atunci când sunt folosiți cu neglijență.

Limbajul C nu include șiruri încorporate, fire, pachete, clase, obiecte, type safety și colectarea gunoiului. Ultima este un dezavantaj major pentru sistemele de operare. Toată stocarea în C este sau statică sau alocare/eliberare explicită de programator, de obicei folosind funcțiile de bibliotecă *malloc* și *free*. Programatorul are controlul total al memoriei - împreună cu pointerii expliciți care îl fac limbajul C atractiv pentru scrierea sistemelor de operare. Sistemele de operare sunt, în esență, sisteme de timp real, chiar și cele cu destinație generală. Când are loc o întrerupere, sistemul de operare poate avea doar câteva microsecunde pentru a efectua unele acțiuni sau pentru a pierde informații critice. Punerea la dispoziție a colectorului de gunoi într-un moment arbitrar este intolerabilă.

1.8.2 Fișiere antet

Un proiect dedicat elaborării unui sistem de operare include, de obicei, câteva directoare, fiecare conținând mai multe fișiere *.c* cu codul pentru o parte a sistemului, împreună cu unele fișiere antet *.h* care conțin declarații și definiții utilizate de unul sau mai multe fișiere de cod. Fișierele antet pot include, de asemenea, **macro**-uri simple, cum ar fi

```
#define BUFFER SIZE 4096
```

care permite programatorului să definească constante, astfel încât atunci când se utilizează BUFFER SIZE în cod, acesta este înlocuit în timpul compilării cu valoarea 4096. O bună practică de programare C este să numești fiecare constantă, cu excepția lui 0, 1 și −1, și uneori chiar acestea. Macro-urile pot avea parametri, cum ar fi

```
#define max(a, b) (a > b? a : b)
```

ceea ce permite programatorului să scrie

```
i = max(j, k+1)
```

și să obțină

```
i = (j > k+1? j : k + 1)
```

pentru a alege valoarea mai mare dintre *j* și *k*+1 în i. Anteturile pot conține, de asemenea, o compilare condiționată, de exemplu

```
#ifdef X86
intel_int_ack();
#endif
```

care se compilează într-un apel către funcția intel ***int_ack*** dacă macro-ul X86 este definit. Compilarea condiționată este folosită pentru a izola codul dependent de arhitectură, astfel încât un anumit cod este inserat doar atunci când sistemul este compilat pe X86, un alt cod este inserat doar atunci când sistemul este compilat pe un SPARC și așa mai departe. Un fișier .c poate include fizic zero sau mai multe fișiere antet folosind directiva **`#include`**. Există mai multe fișiere antet comune pentru aproape toate fișierele *.c* stocate într-un director central.

1.8.3 Proiecte mari de programare

Pentru a construi un sistem de operare, fiecare modul sursă .c este compilat într-un fișier obiect de către compilatorul C. Fișierele obiect, care au sufixul .o, conțin instrucțiuni binare pentru mașina țintă. Ulterior vor fi direct executate de procesor. Nu există nimic ca codul de byte Java sau codul de byte Python în lumea C.

Prima fază de compilare, numită preprocesare C, pe măsură ce citește fiecare fișier .c, de fiecare dată când atinge o directivă #include, procesează fișierul antet din el, extindând macro-urile, gestionând compilarea condiționată (și anumite alte lucruri) și trece rezultatele la următoarea fază de compilare.

Având în vedere că sistemele de operare sunt foarte mari (de obicei, peste cinci milioane de linii de cod), faptul că ar trebui să recompilăm totul de fiecare dată când un fișier este schimbat ar fi insuportabil. Pe de altă parte, modificarea unui fișier antet inclus în mii de alte fișiere necesită recompilarea acelor fișiere. Să urmărești care fișier obiect depinde de care fișier antet este complet imposibil fără ajutor.

Din fericire, calculatoarele sunt foarte bune la acest gen de lucruri. Pe sistemele UNIX, există un program numit *make* (cu numeroase variante precum *gmake*, *pmake*, etc.) care citește *Makefile*, care îi spune ce fișiere depind de alte fișiere. Programul *make* știe ce fișiere obiect sunt necesare pentru a construi codul binar al sistemului de operare și pentru fiecare, verifică dacă vreunul dintre fișierele antet de care depinde fișierul obiect a fost modificat ulterior ultimei compilări. Dacă da, fișierul obiect trebuie recompilat. Doar în acest caz *make* lansează recompilarea, reducând astfel numărul de compilări la minim.

După crearea tuturor fișierelor *.o*, acestea sunt transmise unui program numit *linker* pentru a le combina pe toate într-un singur fișier binar executabil. Funcțiile de bibliotecă apelate sunt, de asemenea, incluse în acest moment, operație numită *legarea externilor*, referințele de funcționare sunt rezolvate și adresele mașinii sunt relocate după caz. Când legarea externilor este încheiată, rezultatul este un program executabil, denumit în mod tradițional *a.out* pe sistemele UNIX. Diferitele componente ale acestui proces sunt ilustrate în fig. 1-30 pentru un

program cu trei fișiere C și două fișiere antet. Deși am discutat despre dezvoltarea sistemului de operare aici, toate acestea se aplică dezvoltării oricărui program mare.

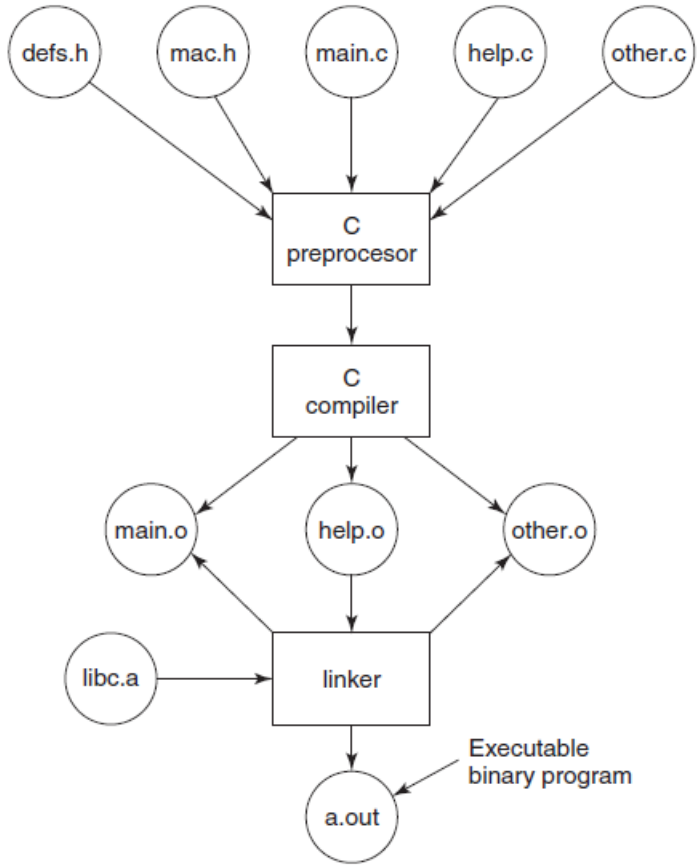

Fig. 1-30. Schema procesului de compilare

După ce modulul executabil a sistemului de operare a fost obținut, computerul poate fi repornit și noul sistem de operare lansat. Odată lansat, acesta poate încărca în mod dinamic componente care nu au fost incluse static în fișierul binar, cum ar fi driverele de dispozitiv și sistemele de fișiere. În timpul funcționării, sistemul de operare poate consta din mai multe segmente: segmentul de cod, de date și stivă. Segmentul de cod este în mod normal imuabil, nu se schimbă în timpul executării. Segmentul de date începe cu o anumită dimensiune și este inițializat cu anumite valori, dar poate crește la nevoie. Stiva este inițial goală, dar crește și se micșorează pe măsură ce funcțiile sunt apelate și returnate. Adesea, segmentul de cod este plasat aproape de începutul memoriei, segmentul de date chiar deasupra acestuia, cu capacitatea de a crește în sus, iar segmentul de stivă la o adresă virtuală mare, cu posibilitatea de a crește în jos.

În toate cazurile, codul sistemului de operare este executat direct de hardware, fără niciun interpretor și fără o compilare just-in-time, așa cum este cazul Java.

1.9 Cercetări în domeniul sistemelor de operare

Informatica este un domeniu care avansează rapid și este greu de făcut prognoze. Cercetătorii universităților și laboratoarelor de cercetare industrială se gândesc în mod constant la idei noi, unele dintre care nu duc nicăieri, dar unele dintre ele devin piatra de temelie a produselor viitoare și au un impact masiv asupra industriei și a utilizatorilor. Separarea grâului de pleavă este deosebit de dificilă, deoarece deseori durează 20-30 de ani de la idee la impact.

De exemplu, când în 1958 președintele Eisenhower a înființat Agenția de Proiecte Avansate de Cercetare (ARPA) a Departamentului Apărării, el pornea de la necesitatea sporirii capacităților de apărare, nici gând să inventeze Internetul. ARPA a finanțat cercetări universitare în cadrul cărora a fost dezvoltat conceptul, nu prea înțeles atunci, de comutare a pachetelor, concept care a condus la prima rețea experimentală de calculatoare cu comutare a pachetelor, ARPANET, pusă în funcțiune în 1969. Multe alte rețele de cercetare finanțate de ARPA au fost conectate la ARPANET și Internetul s-a născut. Internetul a fost apoi folosit cu succes de cercetătorii academici pentru a-și trimite e-mail-uri timp de 20 de ani. La începutul anilor 1990, Tim Berners-Lee de la laboratorul de cercetare CERN din Geneva a inventat World Wide Web, iar Marc Andreesen de la Universitatea din Illinois a scris un browser grafic pentru acesta. Imediat, Internetul s-a umplut de adolescenți pe Twitter. Președintelui Eisenhower nu-i vine să creadă ce a lansat atunci.

Cercetarea în sistemele de operare a dus, de asemenea, la schimbări dramatice în sistemele aplicative. Cum am subliniat anmterior, primele sisteme informatice comerciale erau pentru tratarea pe loturi, până când M.I.T. la începutul anilor 1960, a venit cu idea de partajare a timpului. Interfața utilizatorului era una bazată pe text până când Doug Engelbart a inventat mouse-ul și interfața grafică de utilizator la Stanford Research Institute la sfârșitul anilor 1960. Cine știe ce va urma?

În acest compartiment și în altele asemănătoare, vom arunca o scurtă privire asupra unor cercetări în domeniul sistemelor de operare, doar pentru a oferi o aromă a ceea ce ar putea fi la orizont. Această introducere nu este cu siguranță exhaustică. Se bazează în mare parte pe lucrări care au fost publicate în cadrul conferințelor de cercetare de top, deoarece aceste idei au supraviețuit cel puțin unui proces riguros de evaluare pentru a fi publicate. Rețineți că, în informatică - spre deosebire de alte domenii științifice, majoritatea cercetărilor sunt publicate în conferințe, nu în reviste. Cele mai multe dintre lucrările citate în secțiunile de cercetare au fost publicate de ACM, IEEE Computer Society sau USENIX și sunt disponibile pe internet membrilor acestor organizații. Pentru mai multe informații despre aceste organizații și bibliotecile lor digitale, consultați

ACM	https://www.acm.org/
IEEE Computer Society	https://www.computer.org/
USENIX	https://www.usenix.org/

Practic, toți cercetătorii din domeniul sistemelor de operare își dau seama că sistemele de operare actuale sunt masive, inflexibile, nesigure, nefiabile și pline de bug-uri. În consecință, există o mulțime de cercetări cu privire la modul de construire a unor sisteme de operare mai bune. Au fost publicate lucrări despre bug-uri și depanare (Renzelmann și colab., 2012; și Zhou și colab., 2012), reacția la accidente (Correia și colab., 2012; Ma et al., 2013; Ongaro și colab., 2011; și Yeh și Cheng, 2012), managementul energiei (Pathak și colab., 2012; Petrucci și Loques, 2012; și Shen și colab., 2013), sisteme de fișiere și stocare (Elnably și Wang, 2012; Nightingale și colab., 2012 ; și Zhang și colab., 2013a), I/O de înaltă performanță (De Bruijn și colab., 2011; Li și colab., 2013a; și Rizzo, 2012), hipertratare și multitratare (Liu și colab., 2011), actualizare live (Giuffrida și colab., 2013), gestionarea GPU-urilor (Rossbach și colab., 2011), gestionarea memoriei (Jantz și colab., 2013; și Jeong et al., 2013), sisteme de operare multicore (Baumann și colab., 2009; Kapritsos, 2012; Lachaize și colab., 2012; și Wentzlaff et al., 2012), corectitudinea sistemului de operare (Elphinstone et al., 2007; Yang și colab., 2006; și Klein și colab., 2009), fiabilitatea sistemului (Hruby și colab., 2012; Ryzhyk și colab., 2009, 2011 și Zheng și colab., 2012), confidențialitate și securitate (Dunn și colab., 2012; Giuffrida și

colab., 2012; Li și colab., 2013b; Lorch și colab., 2013; Ortolani și Crispo, 2012; Slowinska și colab., 2012; și Ur și colab., 2012), monitorizarea utilizării și performanței (Harter și colab., 2012; și Ravindranath et al., 2012) și virtualizare (Agesen și colab., 2012; Ben-Yehuda și colab., 2010; Colp și colab., 2011; Dai și colab., 2013; Tarasov și colab., 2013; și Williams și colab., 2012) printre multe alte subiecte.

1.10 Rezumat

Sistemele de operare pot fi privite din două puncte de vedere: ca manager de resurse și mașină extinsă. În viziunea managerului de resurse, sarcina sistemului de operare este de a gestiona în mod eficient diferite părți ale sistemului. Din punctul de vedere al mașinii extinse, fincția sistemului este de a oferi utilizatorilor abstracții care sunt mai convenabile de utilizat decât mașina fizică, abstracții denumite procese, spații de adrese sau fișiere. Sistemele de operare au o istorie lungă, începând din zilele în care au înlocuit operatorul, până la sisteme moderne cu multiprogramare.

Deoarece sistemele de operare interacționează strâns cu hardware-ul, unele cunoștințe despre hardware-ul computerului sunt utile pentru înțelegerea acestora. Calculatoarele sunt construite din procesoare, memorii și dispozitive I/O. Aceste piese sunt conectate prin magistrale de date.

Conceptele în baza cărora sunt construite toate sistemele de operare sunt procesele, managementul memoriei, managementul I/O, sistemul de fișiere și securitatea. Fiecare dintre acestea va fi tratat într-un capitol următor.

Partea principală a oricărui sistem de operare este setul de apeluri de sistem pe care le poate gestiona. Acestea stabilesc ce face de fapt sistemul de operare. Pentru UNIX, am analizat patru grupuri de apeluri de sistem. Primul grup de apeluri de sistem se referă la crearea și terminarea proceselor. Al doilea grup este pentru citirea și scrierea fișierelor. Al treilea grup este destinat managementului directorului. Al patrulea grup conține apeluri diverse.

Sistemele de operare pot fi structurate în mai multe moduri. Mai frecvente structuri sunt sistemele monolitice, sistemele multinivel, microkernel, client-server, mașină virtuală sau exokernel.

Probleme

1. Care sunt cele două funcții principale ale unui sistem de operare?
2. În secțiunea 1.4, sunt descrise nouă tipuri diferite de sisteme de operare. Prezentați o listă de aplicații pentru fiecare din aceste sisteme (una pentru fiecare tip).
3. Care este diferența dintre sistemele cu partajare a timpului și multiprogramarea?
4. Pentru a utiliza memoria cache, memoria principală este împărțită în linii de cache, de obicei 32 sau 64 de octeți. O linie cache este o zonă de memorie tratată ca un tot întreg. Care este avantajul de a memora în cache o linie întreagă în loc de un singur octet sau un cuvânt la un moment dat?
5. Pe computerele timpurii, fiecare octet de date citite sau scrise era gestionat de procesor (adică nu exista DMA). Ce implicații are aceasta pentru multiprogramare?
6. Instrucțiunile legate de accesarea dispozitivelor I/O sunt de obicei instrucțiuni privilegiate, adică sunt executate în modul kernel, nu în modul utilizator. Indicați un motiv pentru care aceste instrucțiuni trebuie să fie privilegiate.
7. Ideea familiei de computere a fost introdusă în anii 1960 pornind cu linia IBM System/360. Această idee este astăzi uitată sau trăiește?
8. Unul dintre motivele pentru care GUI a fost adoptată greu a fost costul hardware-ului necesar pentru susținerea ei. Câtă memorie RAM video este necesară pentru a susține

un ecran monocrom cu 25 de linii × 80 rânduri de caractere? Dar pentru o imagine bitmap culor de 1200 × 900 pixeli pe 24 de biți? Care era fost costul acestei RAM la prețurile din 1980 (5 USD/KB)? Care este prețul de azi?

9. Există mai multe obiective care trebuie urmărite la proiectarea și construirea unui sistem de operare, de exemplu, utilizarea resurselor, actualitatea, robustețea și așa mai departe. Dati un exemplu de doua obiective de proiectare care se pot contrazice.
10. Care este diferența dintre modul kernel și modul utilizator? Explicați modul în care daceste ouă moduri distincte vin în ajutorul dezvoltatorilor de sisteme de operare.
11. Un disc de 255 GB are 65.536 cilindri cu 255 de sectoare pe pistă fiecare sector este de 512 octeți. Câte platouri și capete are acest disc? Presupunând un timp mediu de căutare a cilindrului de 11 ms, o întârziere de rotație medie de 7 msec și o rată de citire de 100 MB/sec, calculați timpul mediu necesar pentru a citi 400 KB dintr-un sector.
12. Care dintre următoarele instrucțiuni ar trebui să fie permise doar în modul kernel?
    - Dezactivați toate întreruperile.
    - Citiți indicațiile ceasului.
    - Setați ceasul.
    - Schimbați harta memoriei.
13. Luați în considerare un sistem care are două CPU, fiecare procesor având două fire (hiperthreading). Să presupunem că trei programe, P0, P1 și P2, sunt pornite cu perioade de rulare de 5, 10 și, respectiv, 20 msec. Cât timp va dura până la finalizarea executării acestor programe? Presupunem că toate cele trei programe sunt 100% legate de procesor, nu se blochează în timpul execuției și nu cedează procesoarele odată alocate.
14. Un computer are o conductă cu patru faze. Fiecare fază are același timp pentru a-și face munca, și anume, 1 nsec. Câte instrucțiuni pe secundă poate executa această mașină?
15. Luați în considerare un sistem de computer care are memorie cache, memorie principală și disc și un sistem de operare care utilizează memorie virtuală. Este nevoie de 1 nsec pentru a accesa un cuvânt din cache, 10 nsec pentru a accesa un cuvânt din memoria RAM și 10 ms pentru a accesa un cuvânt de pe disc. Dacă rata de accesare a memoriei cache este de 95% și rata de accesare a memoriei principale (după un insucces la memoria cache) este de 99%, care este timpul mediu pentru a accesa un cuvânt?
16. Atunci când un program de utilizator efectuează un apel de sistem pentru a citi sau scrie un fișier pe disc, acesta indică numele fișierului, un ppointer către bufferul de date și numărul de simboluri. Controlul este apoi transferat sistemului de operare, care apelează driverul corespunzător. Să presupunem că driverul pornește discul și oprește când apare întreruperea respectivă. În cazul citirii de pe disc, evident că apelantul va trebui să fie blocat (așteaptă datelesolicitate). Dar cazul scrierii pe disc? Este necesar ca apelantul să fie blocat pentru a finaliza transferul de disc?
17. Ce este o instrucțiune capcană (trap)? Explicați utilizarea sa în sistemele de operare.
18. De ce este necesară un tabel al proceselor într-un sistem cu partajarea timpului? Este acest tabel și în sistemele de operare ale calculatoarelor personale care rulează UNIX sau Windows cu un singur utilizator?
19. Există vreun motiv pentru care s-ar putea să doriți să montați un sistem de fișiere într-un director nonempty? Dacă da, care este?
20. Pentru fiecare dintre următoarele apeluri de sistem, dați o condiție care face ca acesta să eșueze: *fork*, *exec* și *unlink*.
21. Ce tip de multiplexare (în timp, în spațiu sau ambele) pot fi utilizate pentru partajarea următoarelor resurse: procesor, memorie, disc, placă de rețea, imprimantă, tastatură și display?

22. Poate oare

```
count = write(fd, buffer, nbytes);
```

atribui lui **count** o altă valoare decât **nbytes**? Dacă da, de ce?

23. Un fișier al cărui descriptor este *fd* conține următoarea secvență de octeți: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5. Sunt efectuate următoarele apeluri de sistem:

```
lseek(fd, 3, SEEK_SET);
read(fd, &buffer, 4);
```

unde apelul **lseek** face o căutare pentru octetul 3 din fișier. Ce conține buferul după ce s-a finalizat citirea?

24. Să presupunem că un fișier de 10 MB este stocat pe un disc pista 50 în sectoare consecutive. Bratul de disc este situat în prezent peste pista 100. Cât timp va dura citirea acestui fișier de pe disc? Presupunem că este nevoie de aproximativ 1 ms pentru a muta brațul de la un cilindru la altul și aproximativ 5 ms pentru a ajunge la sectorul în care începutul fișierului este stocat (rotirea discului). Presupunem că citirea se face cu viteza de 200 MB/s.
25. Care este diferența esențială între un fișier special pentru dispozitivele bloc orientate și un fișier special pentru dispozitivele orientate pe caractere?
26. În exemplul din fig. 1-17, procedura de bibliotecă se numește *read*, iar apelul de sistem este la fel *read*. Este esențial ca ambele să aibă același nume? Dacă nu, care este mai important?
27. Sistemele de operare moderne decuplează spațiul de adrese al procesului de memoria fizică a mașinii. Enumerați două avantaje ale acestui design.
28. Pentru un programator, un apel de sistem arată ca orice alt apel către o procedură de bibliotecă. Este important ca un programator să știe ce proceduri de bibliotecă au ca rezultat apeluri de sistem? În ce circumstanțe și de ce?
29. Figura 1-23 arată că o serie de apeluri de sistem UNIX nu au echivalente Win32 API. Pentru fiecare dintre apelurile enumerate ca neavând un echivalent Win32, care sunt consecințele pentru un programator convertirea unui program UNIX pentru a rula sub Windows?
30. Un sistem de operare portabil este unul care poate fi portat de la o arhitectură de sistem la alta, fără nicio modificare. Explicați de ce nu este fezabil să construiți un sistem de operare complet portabil. Descrieți cele două hi-level necesare pentru proiectarea unui sistem de operare extrem de portabil.
31. Explicați modul în care separarea politicilor și a mecanismelor ajută la construirea sistemelor de operare bazate pe microkernel.