Utilizarea Sistemelor de Operare

Concepte de bază în Sisteme de Operare

- Curs 3 -
20.10.2005

Universitatea POLITEHNICA Bucureşti

2. Concepte în sistemelor de operare

- Există un număr de concepte comune sistemelor de operare
  - procese
  - modul de gestionare a memoriei
  - fişiere

Procese

- Un proces este un program în execuție, căruia i se asociază un spațiu de adrese
- Pentru a putea rula mai multe procese, acestea sunt partajate în timp
  - La reluarea procesului acesta trebuie repornit din starea pe care o avea în momentul suspendării
  - În multe SO, informațiile despre fiecare proces, altele decât conținutul propriului spațiu de adrese, sunt reținute într-un tabel de procese
- Un rol cheie în gestiunea sistemului îl au **apelurile de sistem** pentru inițierea şi terminarea proceselor

Comunicaţia între procese

- Un proces poate crea mai multe procese (numite copii) care la rândul lor pot crea alte procese. Astfel se ajunge la o structura de tip arbore
- Procesele pot comunica între ele. În cazul în care procesul corespondent nu răspunde după un anumit timp, SO trimite un semnal de alarmă care trebuie tratat (de ex. prin retrimiterea mesajului)
- Fiecare utilizator de sistem are asociat un UID (User Identification). Fiecare proces conține UID-ul utilizatorului care l-a declanşat. Un proces copil are acelaşi UID ca şi părintele său.
- Există un UID (superuser în UNIX) care are puteri depline asupra sistemului

2.2 Interblocările

- Când mai multe procese interacționează, pot ajunge la o situație conflictuală din care nu mai pot ieşi. O astfel de situație se numeşte interblocare (deadlock)
- Exemplu:
  - Procesele 1 şi 2 trebuie să copieze date de pe disc pe CD-ROM
  - Procesul 1 solicită şi primeşte acces la disc
  - Procesul 2 solicită şi primeşte acces la CD-ROM
  - Procesul 1 nu primeşte acces la CD-ROM până când procesul 2 nu cedează accesul
  - Procesul 2 nu primeşte acces la disc până când procesul 1 nu cedează accesul

2.3 Gestionarea memoriei

- Fiecare program executat este reținut în memoria principală. Pentru a evita interferențele între ele este necesar un mecanism de protecție
- Fiecare proces are un set de adrese pe care le poate folosi: de la 0 la limita superioară. Limita superioară trebuie să fie mai mică decât dimensiunea memoriei principale pentru a rămâne suficient spațiu pentru SO
- Prin memoria virtuală SO păstrează o parte din spațiul de adrese în memoria principală şi o parte o transferă pe disc, putând realiză transferuri între cele două blocuri de memorie

2.4 Intrare / Ieşire

- Fiecare SO are un subsistem de I/E pentru administrarea dispozitivelor de I/E
- Unele programe sunt independente de dispozitiv, putând lucra cu mai multe dispozitive de I/E
- Există şi drivere specifice unor dispozitive de I/E

2.5 Fişiere

- Sistemul de fişiere ascunde particularităţile discurilor şi ale altor dispozitive de I/E
- Deschiderea, citirea, scrierea şi închiderea unui fişier sunt generate de apeluri de sistem
- Majoritatea SO folosesc conceptul de director (directory) pentru gruparea fişierelor.
  - Intrările dintr-un director pot fi fişiere sau alte directoare.
  - Acest model dă naştere unei ierarhii: sistemul de fişiere
- Ierarhiile de procese nu sunt foarte adânci (în general mai puţin de 3 niveluri). Spre deosebire de acestea, ierarhiile de fişiere sunt mult mai adânci

Identificarea fişierelor

- Fiecare fişier dintr-o ierarhie poate fi identificat prin atribuirea unui nume de cale (path name)
- Calea absolută către un fişier constă dintr-o lista de cataloage care trebuie parcurse pornind de la catalogul rădăcină, care este precedată de separatorul “/”. Componentele căii sunt separate de “/” în UNIX şi “\” în MS-DOS şi Windows
- În fiecare moment un proces are un director de lucru curent în care caută numele de cale care un încep cu “/” (căi relative)
- Pentru a trata dispozitivele de memorie externă care pot fi mutate, în UNIX sistemul de fişiere de pe acestea poate fi adăugat la arborele principal într-un director. Pentru aceasta se foloseşte apelul de sistem *mount*

Drepturi la nivel de fişier

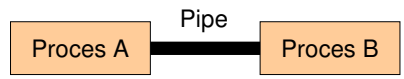
- Înaintea citirii sau scrierii unui fişier, acesta trebuie deschis, moment în care sunt verificate drepturile de acces.
  - Dacă accesul este permis, SO întoarce un număr întreg mic, numit descriptor de fişier.
  - Dacă accesul este interzis, SO întoarce un cod de eroare.

Fişiere speciale

- În UNIX, un concept legat de fişiere este cel de "fişier special". Un fişier special face ca dispozitivele de I/E să fie prezentate utilizatorului sub formă de fişiere. Astfel, pentru lucrul cu un dispozitiv de I/E se folosesc acelaşi apeluri de sistem ca şi pentru lucrul cu fişiere.
- Fişierele speciale sunt păstrate în directorul **`/dev`**
- Fişierele speciale se împart în două categorii
  - Fişiere orientate pe bloc (pentru discuri)
  - Fişiere orientate pe caracter (pentru imprimantă, modem, etc)

Fişiere legătură (pipe)

- O conductă (pipe) este un pseudo-fişier care poate fi folosit pentru conectarea a două procese.



  - Procesul A, pentru a trimite date procesului B, scrie în pipe ca într-un fişier de ieşire.
  - Procesul B, pentru a primi datele, citeşte din pipe ca dintr-un fişier de intrare.

2.6 Securitatea

- SO trebuie să asigure confidențialitatea datelor stocate
- Fişierele din UNIX sunt protejate prin coduri de protecție de 9 biți
  - Un câmp de 3 biți pentru proprietarul fişierului
  - Un câmp de 3 biți pentru ceilalți membri ai grupului proprietarului
  - Un câmp de 3 biți pentru oricine altcineva
  - Fiecare câmp are un bit pentru accesul la citire (**r**ead), unul pentru accesul la scriere (**w**rite) şi unul pentru a permite executarea fişierului (e**x**ecute).
  - Ex: `rwxr-x--x`

2.7 Interpretorul de comenzi

- Interpretorul de comenzi nu face parte din SO
- Interpretorul de comenzi UNIX se numeşte *shell*
- Există mai multe module de interpretare (*sh*, *csh*, *ksh*, *bash*), toate derivând din *sh*
- Interpretorul
  - Vede terminalul ca intrare şi ieşire standard
  - Afişează un prompt atunci când este pregătit să accepte o comandă (de ex. $)
  - Atunci când utilizatorul tastează o comandă (de ex. *date*), este lansat un proces-copil care tratează această comandă

2.7 Interpretorul de comenzi (2)

- Ieşirea standard poate fi redirecționată către un fişier prin plasarea separatorului “>”:

```
date > fişier
```

- Intrarea standard poate fi redirecționată, prin plasarea delimitatorului “<”:

```
date < fişier
```

- Datele de ieşire ale unui program pot constitui date de intrare pentru alt program daca se utilizează o conductă (*pipe*):

```
comandă1 | comandă2
```


3. Apeluri de sistem

- Interfața dintre SO şi programele utilizatorilor este definită de mulțimea apelurilor de sistem.
- Invocarea unui apel de sistem se aseamănă cu invocarea unei proceduri. De exemplu, dacă un proces doreşte să citească date dintr-un fişier:
  - Execută un apel de sistem şi transferă controlul către SO
  - SO deduce ce doreşte sa facă procesul din parametrii apelului de sistem
  - SO execută apelul la nivelul kernelului şi redă controlul procesului, în instrucțiunea care urmează apelului de sistem
- Punerea în corespondență a apelurilor de proceduri POSIX cu apelurile de sistem este aproape unu la unu

Exemplu: apelul sistem de citire

- Apelul sistem *read* poate fi inițiat dintr-un program C prin intermediul procedurii de bibliotecă cu acelaşi nume
- Apelul de sistem *read* are 3 parametri
  - *Primul specifică fişierul*
  - *Al doilea desemnează memoria tampon*
  - *Al treilea specifică numărul de octeți ce trebuie citiți*
- *read* întoarce numărul de octeți citiți efectiv:
  - `cout = read(fd, buffer, nbytes);`
  - Dacă
    - `cout == nbytes`, apelul s-a încheiat cu succes
    - `cout < nbytes` şi `cout != -1`, atunci s-a ajuns la sfârşitul fişierului
    - `cout == -1`, apelul de sistem nu a putut fi executat

read

- Atunci când este apelată funcția de bibliotecă *read* sunt parcurşi următorii paşi:
  - 1, 2 şi 3: programul apelant pune parametrii `fd`, `buffer` şi `nbytes` pe stivă
  - 4: apelul efectiv al procedurii de bibliotecă
  - 5: procedura de bibliotecă pune numărul apelului de sistem într-un registru
  - 6: procedura de bibliotecă execută o instrucțiune TRAP pentru a trece în modul kernel şi a începe execuția de la o adresă fixă din spațiul kernel-ului
  - 7: codul din kernel care începe să se execute citeşte numărul apelului de sistem şi dă controlul rutinei de tratare a apelului de sistem respectiv
  - 8: rulează rutina de tratare a apelului de sistem
  - 9: după ce rutină şi-a terminat treaba, controlul poate fi redat procedurii de bibliotecă la instrucțiunea următoare instrucțiunii TRAP
  - 10: această procedură redă controlul programului apelant
  - 11: programul apelant curăța stiva

Apeluri de sistem pentru gestiunea proceselor

- Pentru administrarea proceselor:
  - pid = fork()
  - pid = waitpid(pid, &statloc, options)
  - s = execve(name, argv, environp)
  - exit(stare)

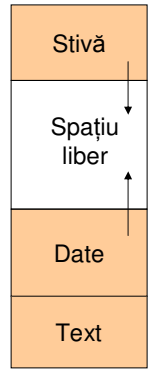
Apeluri de sistem pentru gestiunea fişierelor

- Pentru administrarea fişierelor
  - fd = open(file, how, …)
  - s = close(fd)
  - n = read(fd, buffer, nbytes)
  - n = write(fd, buffer, nbytes)
  - position = lseek(fd, offset, whence)
  - s = stat(name, &buf)

Alte apeluri de sistem

- Pentru administrarea directoarelor şi a sistemelor de fişiere
  - s = mkdir(name, mode)
  - s = rmdir(name)
  - s = link(name1, name2)
  - s = unlink(name1, name2)
  - s = mount(special, name, flag)
  - s = umount(special)
- Diverse
  - s = chdir(dirname)
  - s = chmod(name, mode)
  - seconds = time (&seconds)

Apeluri de sistem pentru gestiunea proceselor



- Procesele în UNIX au spațiul de memorie împărțit în trei segmente: segmentul de text (codul sursă), segmentul de date (variabilele) şi segmentul stivă.
- Segmentul de date creşte de jos în sus, iar cel de stiva de sus în jos.

Apeluri de sistem în sisteme Windows

- În Windows apelurile de bibliotecă şi apelurile de sistem sunt decuplate
- Apelurile de bibliotecă sunt reunite într-un set de proceduri denumit Win32API
- Numărul de apeluri Win32API este de ordinul miilor. Multe dintre ele invocă apeluri de sistem, rulând în mod kernel, dar o mare parte se execută exclusiv în mod utilizator
- În UNIX interfața grafică rulează în întregime în mod utilizator. În WIN32API însă, pentru administrarea acesteia există un număr mare de apeluri de sistem

4. Structura SO

- Cea mai întâlnită formă de organizare
- Nu există o structură clară
- SO este scris ca o colecție de proceduri, fiecare putând să le apeleze pe oricare altă procedură de oricâte ori este nevoie
- Serviciile oferite de SO (apeluri de sistem) sunt cerute punând parametrii într-un loc bine definit şi executând apoi o instrucțiune TRAP.
  - Sistemul trece în mod kernel şi transferă controlul SO.
  - SO preia parametrii şi determină apelul de sistem care trebuie executat.

4.1 Sisteme monolitice (2)

  - Apoi caută într-un tabel o referință la procedura care se ocupă de tratarea apelul de sistem cerut
- Structura de bază a SO este:
  - Un program care invocă procedura de serviciu cerută
  - Un set de proceduri şi servicii care tratează apelurilor de sistem
  - Un set de proceduri utilitare care ajută procedurile de servicii

4.2 Sisteme structurate pe nivele

- SO poate fi construit ca o ierarhie de nivele, fiecare construit deasupra celui de sub el
- Structura SO THE:
  - 5. Operatorul
  - 4. Programe utilizator
  - 3. Managementul I/E
  - 2. Comunicare operator-proces
  - 1. Gestiunea memoriei şi a rolei magnetice
  - 0. Alocarea procesorului şi multiprogramare
- O generalizare şi mai extinsă a conceptului de organizare pe niveluri era prezentă în sistemul MULTICS

4.3 Maşini virtuale (1)

- Primele versiuni de OS/360 erau sisteme de prelucrare pe loturi. Pentru acestea a apărut un sistem care să permită rularea cu divizarea timpului. Acest sistem se numeşte VM/360 şi funcționează pe principiul:
  - Un sistem cu divizarea timpului oferă: multiprogramare şi o maşina extinsă cu o interfață mai comodă decât hardware-un simplu
- Partea principală a sistemului, monitorul maşinii virtuale, rulează direct pe hardware şi realizează multiprogramarea, oferind mai multe maşini virtuale nivelului superior

4.3 Maşini virtuale (2)

- Maşinile virtuale sunt copii exacte ale hardware-ului, incluzând modurile kernel şi utilizator, I/E, întreruperi, etc. Prin urmare ele pot rula orice SO care ar rula direct pe hard
- Unele maşini virtuale rulează un SO monoutilizator, interactiv (CMS: Conventional Monitor System). CMS trimite apelul de sistem către SO virtual, generând apoi instrucțiuni de I/E normale pentru citirea discului său virtual
- Maşinile virtuale sunt folosite azi pentru a rula programe MS-DOS pe 16 biți pe un Pentium
  - Instrucțiunile normale rulează direct pe hard

4.3 Maşini virtuale (3)

  - Atunci când se execută o instrucțiune TRAP către SO pentru a face un apel de sistem, această instrucțiune ajunge la monitorul maşinii virtuale
- Alt domeniu în care sunt folosite maşinile virtuale este rularea programelor Java
  - Sun Microsystems a introdus JVM (Java Virtual Machine)
  - Compilatorul de Java produce cod pentru JVM, care este executat de un interpretor virtual JVM
  - Avantajul este dat de faptul că programele rulează pe calculatorul client şi nu pe server

4.4 Exokernel-uri

- SO oferă fiecărui utilizator într-o maşină virtuală o clonă a sistemului respectiv, dar având doar un subset de resurse (de ex., doar adresele de memorie între 0 şi 1023)
- La nivelul cel mai de jos se afla un program numit exokernel, care rulează în mod kernel. El oferă resurse maşinilor virtuale şi verifică respectarea acestora
- În acest fel dispare nivelul de punere în corespondență

4.5 Modelul client-server (1)

- În SO moderne există tendința de a muta codul la nivelele superioare li de a scoate cât mai mult posibil din modul kernel, rămânând un microkernel minimal
- În general pentru aceasta se implementează cea mai mare parte a SO în procese utilizator
- Pentru a cere un serviciu, un proces utilizator (proces client) trimite o cerere unui proces server, care o procesează şi trimite înapoi răspunsul
- Este de dorit ca procesul server să ruleze în mod utilizator
- Kernel-ul doar gestionează comunicația între clienți şi servere

4.5 Modelul client-server (2)

- Avantaje:
  - Prin divizarea SO, acesta devine mai uşor de administrat
  - Dacă un serviciu se blochează, întreg SO nu va face acest lucru
  - Modelul client-server poate fi adaptat pentru a putea fi folosit în sisteme distribuite