Sisteme de Operare
note de curs

March 8, 2014

Chapter 1

Introducere

Chapter 2

Folosirea Sistemelor de Fisiere

Suport curs: OSC - Capitolul 10 si 11.1, MOS - Sectiunile 6.1 si 6.2

Memoria RAM este volatila si nu poate stoca toate datele necesare tuturor aplicatiilor. Exista necesitatea de a stoca datele pe un mediu durabil, si distingem 3 cerinte:

- Trebuie sa putem stoca o cantitate foarte mare de informatie
- Informatia trebuie sa existe si dupa ce procesul care a creat-o sau o foloseste se termina.
- Mai multe procese trebuie sa poata accesa informatia simultan.

Discurile magnetice sunt suportul predilect de stocare a informatiei. Desi alte solutii exista aceste solutii sunt fie prea lente (e.g. banda magnetica) fie prea scumpe inca (SSD). Pentru acest curs, este suficient sa spunem despre discurile magnetice ca ofera un API pentru a scrie si citi un bloc de date.

Desi folosirea acestui API poate rezolva in principiu problema stocarii, totusi el este anevoios pentru utilizatori. Orice utilizator al unui disk trebuie sa se asigure ca stie ce blocuri sunt libere, ca poate gasi informatia stocata si ca utilizatori diferiti nu isi pot citi datele decat cu permisiune.

2.1 Fisiere

Pentru a rezolva aceasta problema, vom folosi o abstractie: **fisierul**- o unitate de informatie creata de un proces. Partea din SO care se ocupa cu managementul fisierelor se cheama sistem de fisiere.

In acest curs vom discuta interfata oferita de sistemul de fisiere utilizatorilor.

Cand un proces creaza un fisier, ii da un nume. Fisierul continua sa existe si dupa ce procesul se termina. Multe SO (e.g. Windows) formeaza numele unui

fisier din doua parti separate de "."; partea de dupa "." se numeste extensie si indica tipul fisierului. Sistemele Unix pot numi fisierele astfel, insa extensiile sunt doar conventii si nu sunt interpretate SO in nici un fel.

Informatia dintr-un fisier poate fi organizata ca o secventa de octeti sau o insiruire de inregistrari (record). Prima varianta este folosita pe scara larga pentru ca permite aplicatiilor sa isi defineasca orice tip de structura intr-un fisier.

Fisierele sunt de mai multe tipuri: fisiere normale (contin informatia de la utilizatori), directoare (structureaza sistemul de fisiere, contin alte fisiere si directoare). Exista si fisiere speciale caracter (pentru a comunica cu dispozitivele seriale) si block (pentru a comunica cu un hard disk).

Fisierele simple sunt ASCII sau binare. Fisierele ASCII contin linii de text terminate de CR sau CR/LF si au marele avantaj de a fi usor de citit. Fisierele binare sunt fisiere non-ASCII - daca le afisam vom vedea niste caractere incomprehensibile! Fisierele binare au o structura cunoscuta programelor care le-au creat.

Initial fisierele erau accesate secvential (legacy caseta!); discurile magnetice fac posibila citirea in orice ordine, ducand la aparitia fisierelor cu access aleator. Aceste fisiere au asociat un cursor de fisier care este 0 atunci cand fisierul este deschis, si incrementat la fiecare citire sau scriere. Acest cursor poate fi repozitionat cu comanda seek (Windows, Unix).

Fiecare fisier are asociat un nume si datele sale, si o multime de alte atribute numite metadate cum ar fi: dimensiunea fisierului, timp de creare, timp ultimei modificari, etc. Aceste metadate variaza de la system la sistem, insa includ in general un identificator al persoanei care a creat fisierul si permisiunile pentru fisierul respectiv.

Operatii pe fisiere:

- Creare de fisier. Creaza un fisier fara date, seteaza atribute initiale.
- Sterge fisier.
- Deschide: permite SO sa aduca in memorie metadatele fisierului pentru acces rapid. Deasemenea permite procesului sa foloseasca un "file handle" pentru a numi fisierul dupa ce l-a deschis, iar asta permite manipularea fisierului (e.g. redenumirea lui) fara a fi nevoie sa se schimbe si handle-ul respectiv.
- Inchide: anunta SO ca fisierul nu mai e folosit, sunt salvate datele din fisier (din ultimul block, chiar daca nu este plin).
- Citeste / scrie / repozitioneaza.
- Seteaza / citeste atribute.

- Redenumeste.

Fiecarui proces sistemul de operare ii asociaza o tabela de descriptori care va inregistra toate fisierele deschise de acel proces, consolele, etc. Comanda *lsof* afiseaza tabela de descriptori aferenta unui proces, inclusiv offset-ul current pentru fisierele deschise.

Primii trei descriptori au o semnificatie speciala (0-standard input, 1-standard output, 2-standard error). In programele demon aceste intrari indica /DEV/NULL.

Atunci cand dorim sa redirectam intrarea/iesirea unui program, o putem face fie din linia de comanda (exemplu ./APP ¿ FISIER_IESIRE ¡ FISIER_INTRARE) sau din programatic folosind apelul POSIX DUP-vezi demo pentru exemplu de utilizare.

API-ul de fisiere se poate face fie folosind direct suportul sistemului de operare (POSIX: read, write, open, close, etc) sau folosind implementarea in biblioteca LIBC a standardului ANSI C (fread, fwrite, fopen, fclose). Diferenta majora intre aceste doua optiuni este folosirea bufferelor in varianta ANSI: scrierea sau citirea se va executa dintr-un buffer in memoria utilizator a bibliotecii, care la randul ei va folosi API-ul POSIX pentru a interactiona (mai rar) cu sistemul de operare. Demo-ul asociat va arata aceste diferente in mod practic: folosirea API-ului ANSI C duce la performante mai bune in general pentru ca numarul de apeluri de sistem este redus, dar costul platit este intarzierea scrierii datelor pana cand biblioteca hotaraste ca este cazul sa scrie, sau pana cand fflush este apelat.

2.2 Directoare

Directoarele structureaza informatia din sistemul de fisiere. Ele sunt de multe ori implementate cu fisiere, cu un format special. Exista mai multe tipuri de sisteme de directoare: un singur nivel (care contine toate fisierele) si ierarhic care permite crearea unui arbore de directoare.

Pentru a numi un fisier intr-un sistem de fisiere ierarhic se foloseste **calea absoluta** de la directorul radacina, sau o **calea relativa** de la directorul curent. Fiecare proces are directorul sau curent; din acest motiv bibliotecile nu schimba directorul curent - ar afecta procesele care presupun ca directorul curent este constant.

Fiecare director in afara de "/" are doua intrari speciale: ".." si "." care se refera la directorul parinte si directorul curent, respectiv. Operatii pe directoare:

- Creaza un director gol, care are doar intrarile "." si "..".
- Sterge un director doar daca este gol.
- Deschide un director pentru a citi lista sa de fisiere.

- Inchide un director
- Citeste director: intoarce urmatorul fisier din director
- Redenumeste
- Creaza o legatura, permitand unui fisier sa apara in mai multe directoare simultan.
- Sterge legatura.

Chapter 3

Procese

Suport curs: OSC - Capitolul 3, MOS - Capitolul 2.1

Programatorii scriu programe care citesc date de intrare, le proceseaza si produc rezultate necesare utilizatorilor. Programele sunt in general stocate pe medii durabile (discuri magnetice, SSD). Inainte de a fi executate programele trebuie incarcate in memorie, iar abia dupa aceea functia principala a programelor paote fi executata (functia main). Executia unui program inceteaza atunci cand functia main se termina, sau cand executia este terminata explicit prin apelul functiei exit sau prin transmiterea unui semnal catre acel program in executie.

La un moment dat, executia unui program necesita accesul exclusiv la resursele calculatorului (exemplu procesor, disc, etc). Executia unor programe dureaza mult timp, si pentru ca acestea asteapta date de la dispozitivele de intrare/iesire—astfel, un sistem de operare care ar executa un singur program simultan ar folosi resursele intr-un mod foarte ineficient. Este necesar ca mai multe programe sa poate fi executate simultan pentru a creste productivitatea sistemelor de calcul.

Pentru a permite acest lucru, sistemele de operare ofera o abstractie fundamentala numita proces. Procesul abstractizeaza un program in executie, capturand resursele folosite de program si starea lor. Procesul da iluzia programului ca ocupa exclusiv procesorul: un proces executa in mod secvential instructiunile unui program (sau alei unui thread de executie din cadrul unui program - vezi cursul 5).

Urmatoarea paralela adaptata dupa Tanenbaum surprinde foarte bine diferenta dintre program si proces. Programul este asemanator unei retete, este o entitate statica. Procesorul este bucatarul care interpreteaza reteta, folosind cateva intrari (ingrediente) si producand rezultate (feluri de mancare). In timp ce un bucatar asteapta ca apa de la ciorba sa fiarba (i.e. atunci cand un program asteapta dupa I/O), el poate incepe sa lucreze la un alt fel de mancare, cu o alta reteta, pentru a isi creste productivitatea. Pentru a lucra la mai multe feluri de mancare simultan, bucatarul trebuie sa tina minte starea ciorbei (i.e. care sunt urmatorii pasi din reteta) inainte sa se apuce de celalalt fel de mancare.

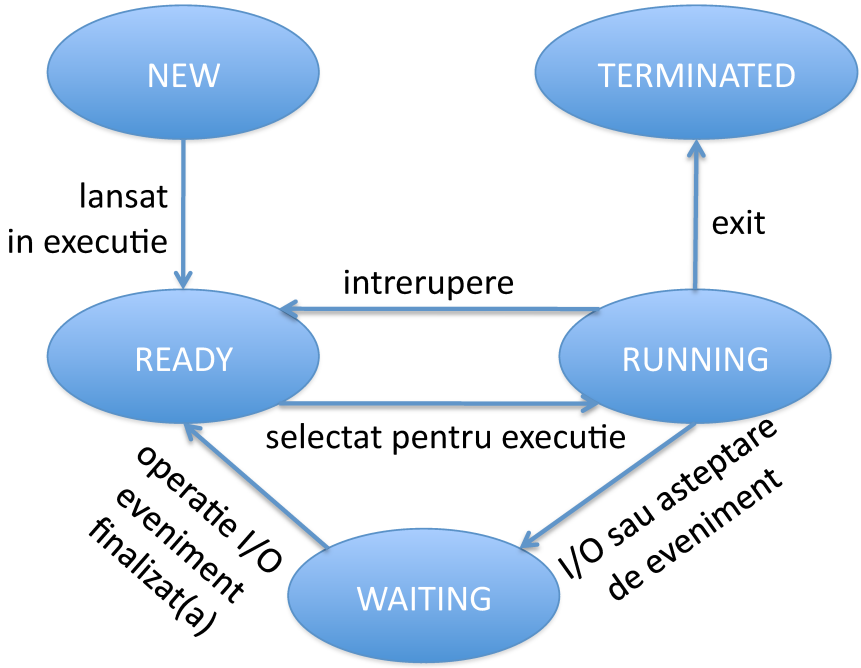

Figure 3.1: Diagrama de stari a unui proces

Procesul tine evidenta resurselor folosite de program, inclusiv a fisierelor (stocare) si retelei (sockets). Fiecare astfel de resursa are un identificator unic (descriptor) si o stare asociata. Aceste resurse pot fi accesate unic de procesul in discutie, sau pot fi partajate cu alte procese.

Un proces poate avea mai multe stari, asa cum arata diagrama din Figura 3.1

Ierarhia de procese Fiecare proces are un identificator unic in sistemul de operare numite process ID (PID). In sistemele Unix, fiecare proces are un proces parinte, iar procesele formeaza o ierarhie de procese. Identificatorul procesului parinte se numeste ppid (sau parent PID). Pentru a visualisa ierarhia de procese in Linux se poate folosi utilitarul PSTREE.

Planificarea proceselor Un SO executa mai multe procese in acelasi timp, medind accesul la resursele calculatorului pentru aceste procese si asigurand ca nici un proces nu monopolizeaza aceste resurse un timp indelungat. In practica, numarul de procese care ruleaza simultan este mai mic sau egal cu numarul de procesoare distincte din sistemul de calcul. Sistemul de operare da iluzia executiei simultane a mai multor procese pe acelasi procesor schimband de multe ori pe secunda procesul care are acces la procesor.

Cuanta de timp este un parametru al SO care decide cat poate un procesor rula ne-intrerupt, si variaza intre 0.1ms pana la 100ms. O cuanta mai mica asigura o granularitate mai buna a impartirii resurselor, insa rezulta intr-un cost mai mare platit pentru schimbarile rapide de context intre procese.

Un proces in executie ruleaza pana cand:

- Ii expira cuanta de timp alocata. In acest caz procesul va fi pus in starea "ready" si va astepta sa fie planificat din nou.
- Asteapta dupa un apel de I/O - procesul va fi trecut in coada "waiting" pentru resursa respectiva. Atunci cand cererea este finalizata, procesul revine in starea "ready"
- Se termina

Crearea unui proces Pentru a crea un proces, un proces existent va folosi un apel de sistem. Procesul nou are la baza un executabil, caruia i se asocieaza resurse cu ajutorul loader-ului (ld-linux.so in Linux). La intrare se primeste programul executabil si argumentele, iar la iesire va aparea un nou proces, cu un nou PID, si o tabela de descriptori noua.

In Unix, procesul existent va apela functia FORK care va crea un proces copil identic cu procesul parinte, si care partajeaza multe resurse (e.g. descriptorii de fisier). EXEC transforma procesul copil pornind de la un executabil si invocand loader-ul, resursele fiind unice fiecarui proces, procesul transformat avand alt spatiu de adresa. PID-ul nu se schimba.

In Windows, nu exista doua apeluri de sistem separate ca in Unix. Functia CreateProcess indeplineste ambele functionalitati, si de aceea are multi parametri.

API-ul pentru a crea procese depinde de mediul folosit. In shell, in mod automat se creaza procese noi atunci cand utilizatorul doreste sa execute diferite utilitare (e.g. ls, ps). API-ul ANSI C include apelul SYSTEM care creaza un nou proces si executa acolo programul specificat impreuna cu datele de intrare; acest API e suportat de majoritatea SO existent. In fine, familia de functii popen/pclose/pread/pwrite creaza un proces nou si permit procesului existent sa comunice cu procesul nou creat cu ajutorul a doua pipe-uri, conectate la intrarea si iesirea standard a noului proces. Acest API e deasemenea suportat de SO Windows, desi nu este in standardul ANSI.

Un proces se incheie atunci cand functia main isi termina executia, se apeleaza exit sau se primeste un semnal. Procesul returneaza o valoare parintelui sau (valoare de retur).

Pentru a primi aceste informatii remanente dupa terminarea procesului, procesul parinte (sau alt proces in Windows) poate "astepta" terminarea procesului.

Aceste functii iau ca intrare PID-ul procesului asteptat, si intorc valoarea de retur a procesului. Asteptarea este o forma de sincronizare inter-proces.

In Unix exista o ierarhie stricta de procese si in mod implicit procesul parinte trebuie sa astepte procesul copil sa termine. Apar doua situatii exceptionale: atunci cand parintele termina inaintea copilului, se spune ca procesul copil este orfan; rolul parintelui va fi preluat automat de procesul init, cel care sta la baza ierarhiei de procese in Unix. Al doilea caz este atunci cand procesul parinte exista dar nu executa functia de asteptare pentru copil in momentul cand copilul isi termina executia. In acest caz copilul devine "zombie" si va ramane in SO pana cand procesul parinte executa WAIT.

3.1 Comunicarea interproces

Procesele pot comunica folosind mai multe mecanisme distincte.

Semnalele sunt un mecanism de IPC specific Unix si permit unui proces sau nucleului SO sa notifice procesul apelat de existenta unei situatii neprevazute. Semnalele sunt identificate printr-un numar si sunt trimise folosind apelul de sistem kill. Semnalele sunt tratate de catre handlere inregistrate de procese. Handlerul default termina procesul.

Pipe-urile sunt canale de comunicatie unidirectionale care pot fi folosite de procesele inrudite. Un pipe are asociati doi descriptori: unul de scriere, si unul de citire. In Unix, apelul fork partajeaza descriptorii de fisier (si implicit de pipes), permitand proceselor copil/parinte sa comunice foarte usor: un proces scrie iar celalalt citeste. Exista pipe-uri cu nume (cele care au intrare in sistemul de fisiere) si pipe-uri anonime. Pipe-urile sunt folosite atunci cand se foloseste operator "—" in linia de comanda. Pipe-urile cu nume permit comunicarea intre procese care nu sunt inrudite.

Memoria partajata este o modalitate foarte eficienta de comunicare inter-proces: un proces declara o zona de memorie partajata, iar alte procese se pot "atasa" la aceasta memorie, avand dupa aceea dreptul de a scrie si a citi direct, fara a folosi apeluri de sistem. Lucrul cu memorie partajata necesita sincronizare explicita intre programe.

Socket-ii sunt cel mai folosit mecanism pentru comunicare inter-proces; ei vor fi discutati in detaliu in cursurile urmatoare.