**Tema Casa Sisteme de Operare**

**Comparatie intre implemenatarea**

**firelor de executie si**

**Proceselor la Windows si Linux**

**Stancescu Cristian Gabriel**

**Grupa 431 A**

**CUPRINS**

1 INTRODUCERE………………………………………………………………………………………………….3

1.1 PROCESE………………………………………………………………………………………………………3

1.2 Diferente dintre thread-uri şi procese…………………………………………………………………...4

1.3Avantajele thread-urilor…………………………………………………………………………………….5

1.4 Tipuri de Thread-uri…………………………………………………………………………………………5

1.5 Atributele unui thread……………………………………………………………………………………….6

1.6 KERNELUL……………………………………………………………………………………………………..6

2 LINUX………………………………………………………………………………………………………………7

2.1 API………………………………………………………………………………………………………………..7

2.2 Suport POSIX………………………………………………………………………………………………….7

2.3 Procese in LINUX……………………………………………………………………………………………..8

2.3.1 Creearea unui process…………………………………………………………………………………….8

2.4 Creeare firelor de executie………………………………………………………………………………….9

2.4.1 Asteptarea firelor de executie……………………………………………………………………………9

2.4.2 Terminarea firelor de executie…………………………………………………………………………..10

2.5 Categorii de thread-uri………………………………………………………………………………………10

2.6 Atributele unui thread……………………………………………………………………………………..11

3 Windows…………………………………………………………………………………………………………11

3.1 API………………………………………………………………………………………………………………11

3.2 Creearea unui process……………………………………………………………………………………..12

3.3 Creearea firelor de executie……………………………………………………………………………….13

3.3.1 Asteptarea firelor de executie………………………………………………………………………….14

3.3.2 Terminarea firelor de executie…………………………………………………………………………14

3.3.3 Suspend, Resume………………………………………………………………………………………..15

3.4 Fibre de executie…………………………………………………………………………………………….15

3.5 Concluzie………………………………………………………………………………………………………15

**1 Introducere**

Un fir de execuţie există în cadrul unui proces, i reprezintă o unitate de execuţie mai fină decât acesta. În momentul în care un proces este creat, în cadrul lui există un singur fir de execuţie, care execută programul secvenţial. Acest fir poate la rândul lui sa creeze alte fire de execuţie; aceste fire vor rula porţiuni ale binarului asociat cu procesul curent, posibil aceleaşi cu firul iniţial (care le-a creat)

**1.1 PROCESE**

Un proces este un program în execuție. Procesele sunt unitatea primitivă prin care sistemul de operare alocă resurse utilizatorilor. Orice proces are un spațiu de adrese și unul sau mai multe fire de execuție. Putem avea mai multe procese ce execută același program, dar oricare două procese sunt complet independente.Informațiile despre procese sunt ținute într-o structură numită Process Control Block (PCB), câte una pentru fiecare proces existent în sistem. Printre cele mai importante informații regăsim:

* PID - identificatorul procesului
* spațiu de adresă
* registre generale, PC (contor program), SP (indicator stivă)
* tabela de fișiere deschise
* informații referitoare la semnale
* lista de semnale blocate, ignorate sau care așteaptă să fie trimise procesului
* handler-ele de semnale
* informațiile referitoare la sistemele de fișiere (directorul rădăcină, directorul curent)

În momentul lansării în execuție a unui program, în sistemul de operare se va crea un proces pentru alocarea resurselor necesare rulării programului respectiv. Fiecare sistem de operare pune la dispoziție apeluri de sistem pentru lucrul cu procese: creare, terminare, așteptarea terminării. Totodată există apeluri pentru duplicarea descriptorilor de resurse între procese, ori închiderea acestor descriptori.

Procesele pot avea o organizare:

* ierarhică - de exemplu pe Linux - există o structură arborescentă în care rădăcina este procesul init (pid = 1).
* neierarhică - de exemplu pe Windows.

În general, un proces rulează într-un mediu specificat printr-un set de variabile de mediu. O variabilă de mediu este o pereche NUME = valoare. Un proces poate să verifice sau să seteze valoarea unei variabile de mediu printr-o serie de apeluri de bibliotecă. ( Linux, Windows )

Pipe-urile (canalele de comunicație) sunt mecanisme primitive de comunicare între procese. Un pipe poate conține o cantitate limitată de date. Accesul la aceste date este de tip FIFO (datele se scriu la un capăt al pipe-ului pentru a fi citite de la celălalt capăt). Sistemul de operare garantează sincronizarea între operațiile de citire și scriere la cele două capete. ( Linux, Windows )

Există două tipuri de pipe-uri:

* pipe-uri anonime: pot fi folosite doar de procese înrudite (un proces părinte și un copil sau doi copii), deoarece sunt accesibile doar prin moștenire. Aceste pipe-uri nu mai există după ce procesele și-au terminat execuția.
* pipe-uri cu nume: au suport fizic - există ca fișiere cu drepturi de acces. Aceasta înseamnă că ele vor exista independent de procesul care le creează și pot fi folosite de procese neînrudite.

**1.2 Diferente dintre thread-uri şi procese**

* procesele nu partajează resurse între ele (decât dacă programatorul foloseşte un mecanism special pentru asta - vezi IPC), pe când thread-urile partajează în mod implicit majoritatea resurselor unui proces. Modificarea unei astfel de resurse dintr-un fir este vizibilă instantaneu şi celorlalte:
* segmentele de memorie precum .heap, .data şi .bss (deci şi variabilele stocate în ele)
* descriptorii de fişiere (aşadar, închiderea unui fişier este vizibilă imediat pentru toate thread-urile)
* sockeţii
* fiecare fir are un context de execuţie propriu, format din:
	+ - stivă
		- set de regiştri (deci şi un contor de program - registrul (E)IP)

Procesele sunt folosite de SO pentru a grupa şi aloca resurse, iar firele de execuţie pentru a planifica execuţia de cod care accesează (în mod partajat) aceste resurse.

**1.3 Avantajele thread-urilor**

Deoarece thread-urile aceluiaşi proces folosesc toate spaţiul de adrese al procesului de care aparţin, folosirea lor are o serie de avantaje:

* crearea/distrugerea unui thread durează mai puţin decât crearea/distrugerea unui process
* timpul context switch-ului între thread-urile aceluiaşi proces este foarte mic, întrucât nu e necesar să se "comute" şi spaţiul de adrese (pentru mai multe informaţii, căutaţi "TLB flush" pe google)
* comunicarea între thread-uri are un overhead minim (practic se face prin modificarea unor zone de memorie din spaţiul de adresă)

Firele de execuţie se pot dovedi utile în multe situaţii, de exemplu, pentru a îmbunătăţi timpul de răspuns al aplicaţiilor cu interfeţe grafice (GUI), unde prelucrările CPU-intensive se fac de obicei într-un thread diferit de cel care afişează interfaţa. De asemenea, ele simplifică structura unui program i conduc la utilizarea unui număr mai mic de resurse (pentru că nu mai este nevoie de diversele forme de IPC pentru a comunica).

**1.4 Tipuri de thread-uri**

Există 3 categorii de thread-uri :

* **Kernel Level Threads (KLT)**
* **User Level Threads (ULT)**
* **Fire de execuţie hibride**

**1.4.1 Kernel Level Threads.** Managementul thread-urilor este făcut de kernel, şi programele user-space pot crea/distruge thread-uri printr-un set de apeluri de sistem. Kernel-ul menţine informaţii de context atât pentru procese cât i pentru thread-urile din cadrul proceselor, iar planificarea pentru execuţie se face la nivel de thread**.**

**1.4.2 User Level Threads.** Kernel-ul nu este contient de existenţa lor, şi managementul lor este făcut de procesul în care ele există, folosind de obicei o bibliotecă. Astfel, schimbarea contextului nu necesită intervenţia kernel-ului, iar algoritmul de planificare depinde de aplicaţie.

**1.4.3 Fire de execuţie hibride.**  Aceste fire încearcă să combine avantajele thread-urilor user-level cu cele ale thread-urilor kernel-level. O modalitate de a face acest lucru este de a utiliza fire kernel-level pe care să fie multiplexate fire user-level. KLT sunt unităţile elementare care pot fi distribuite pe procesoare. De regulă crearea thread-urilor se face în user space i tot aici se face aproape toată planificarea şi sincronizarea. Kernel-ul ştie doar de KLT-urile pe care sunt multiplexate ULT, şi doar pe acestea le planifică. Programatorul poate schimba eventual numărul de KLT alocate unui proces.

**1.5 Atributele unui thread**

Atributele reprezintă o modalitate de specificare a unui comportament diferit de comportamentul implicit. Atunci când un fir de execuţie este creat cu pthread\_create se poate specifica un atribut pentru respectivul fir de execuţie. Atributele implicite sunt suficiente pentru marea majoritate a aplicaţiilor. Cu Funcţii pentru cleanup 7 ajutorul unui atribut se pot schimba:

* starea: unificabil sau detaşabil
* politica de alocare a procesorului pentru thread-ul respectiv (round robin, FIFO, sau system default)
* prioritatea (cele cu prioritate mai mare vor fi planificate, în medie, mai des)
* dimensiunea şi adresa de start a stivei

**1.6 KERNELUL**

Kernelul reprezintă componenta centrală a sistemelor de operare. Este reponsabil pentru administrarea resurselor sistemului şi pentru administrarea accesului programelor la aceste resurse. Exista mai multe tipuri de kernel: monolitice, hibride, microkerneluri, nanokerneluri şi exokerneluri.

**2 LINUX**

**2.1 API**

 In general prin API (Application Programming Interface) al unui sistem, se intelege o interfata definita printr-un set de functii, ce este disponibila unui programator pentru a interactiona cu acel sistem. Aceasta interfata este caracteristica unui sistem de operare si unui limbaj de programare.
Pentru dezvoltarea de aplicatii distribuite in reteaua Internet, sistemul de operare Linux ofera programatorilor o interfata API Internet, cu sistemul de comunicatie, dupa modelul Berkeley Socket, concept introdus pentru prima data in anul 1981, odata cu versiunea BSD 4.1 UNIX. Alte API Internet: winsock, java, System V TLI (Transport Layer Interface), Novell Netware API. In continuare vom prezenta Internet API dupa modelul Berkeley socket, pentru limbajul C in Linux, astfel, incat la sfarsitul acestui material programatorul va putea aborda orice problema in acest domeniu.

In Linux sistemul de comunicatie este  inclus in nucleul sistemului de operare si este organizat sub forma unei stive de protocoale, numita stiva TCP/IP.   In analogie cu Modelul OSI de Referinta, apelurile (functii) de sistem definite de Internet API reprezinta primitivele de serviciu la interfata cu stiva de protocoale. Prin parametrizarea apelurilor de sistem, Linux ofera programatorilor interfete uniforme catre  protocoalele de comunicatie : TCP, UDP, IP, ICMP.

**2. 2 Suport POSIX**

În ceea ce priveste thread-urile, POSIX nu specifica dacă acestea trebuie implementate în user-space sau kernel-space. Linux le implementează în kernel-space, dar nu diferenţiază thread-urile de procese decât prin faptul că thread-urile partajează spaţiul de adresă (atât thread-urile, cât şi procesele, sunt un caz particular de "task"). Pentru folosirea thread-urilor în Linux trebuie să includem header-ul pthread.h unde se găsesc declaraţiile funcţiilor i tipurilor de date necesare şi să utilizăm biblioteca libpthread.

**2.3 PROCESE IN LINUX**

Lansarea in execuție a unui program presupune urmatorii pasi:

* Se creează un nou proces cu fork - procesul copil are o copie a resurselor procesului parinte.
* Dacă se dorește înlocuirea imaginii procesului copil aceasta poate fi schimbata prin apelarea unei functii din familia exec.

**2.3.1 Creearea unui proces**

Creearea unui proces În Linux singurul mod prin care se poate creea un nou proces este prin apelul de sistem fork. Efectul acestui apel este creearea unui nou proces, copie a procesului care a apelat fork. Difera doar PID-ul proceselor, noul proces primind un alt PID de la sistemul de operare. La pornirea sistemului in modul nucleu,exista un singur proces initial parinte. Dupa ce sistemul a fost initializat acest proces parinte lanseaza un fir de executie(un proces copil) numit **init** dupa care ramane inactiv.Structura task\_struct alocata procesului parinte este singura care nu este alocata dinamic ci este declarata ca variabila statica in codul sursa al kernelului(se numeste *init\_task*). Dupa ce a fost lansat in executie,procesul **init** initializeaza sistemul (initializeaza consola, monteaza sistemul principal de fisiere) dupa care,in functie de continutul fisierului /etc/inittab lanseaza in executie alte procese. Din aceasta faza noi procese pot fi create prin clonarea celor deja existente prin apelarea unor proceduri de sistem(clone,fork,vfork)

****

**2.4 Crearea firelor de execuţie**

Crearea firelor de execuţie Pentru crearea unui nou fir de execuţie se foloseste funcţia pthread\_create :

 #include <pthread.h>

 int pthread \_create(pthread\_t \*tid, const pthread\_attr\_t \*tattr, void\*(\*start\_routine)(void \*), void \*arg);

Noul fir creat se va executa concurent cu firul de execuţie din care a fost creat. Acesta va executa codul specificat de funcţia start\_routine căreia i se va pasa argumentul arg. Folosind arg se poate transmite firului de execuţie un pointer la o structură care sa conţină toţi "parametrii" necesari acestuia. Prin parametrul tattr se stabilesc atributele noului fir de execuţie. Dacă transmitem valoarea NULL thread-ul va fi creat cu atributele implicite.

**2.4.1 Asteptarea firelor de execuţie**

La fel ca la procese, un părinte îi poate atepta fiul apelând pthread\_join (înlocuiete waitpid).

int pthread\_join(pthread\_t th, void \*\*thread\_return);

Primul parametru specifică identificatorul firului de execuţie ateptat, iar al doilea parametru specifică unde se va plasa codul întors de funcţia copil (printr-un pthread\_exit sau printr-un return). În caz de succes se întoarce valoarea 0, altfel se întoarce o valoare negativă reprezentând un cod de eroare.

**2.4.2 Terminare firelor de executie**

Un fir de execuţie se termină la un apel al funcţiei pthread\_exit :

void pthread\_exit(void \*retval);

Dacă nu există un astfel de apel este adăugat unul, în mod automat, la sfâritul codului firului de execuţie. Prin parametrul retval se comunică părintelui un mesaj despre modul de terminare al copilului. Această valoare va fi preluată de funcţia pthread\_join. Metodele ca un fir de execuţie să termine un alt thread sunt: stabilirea unui protocol de terminare (spre exemplu, firul master setează o variabilă globală, pe care firul slave o verifică periodic). • mecanismul de "thread cancellation", pus la dispozitie de libpthread. Totuşi, această metodă nu este recomandată, pentru că este greoaie, şi pune probleme foarte delicate la clean-up. Pentru mai multe detalii, consultaţi urmatorul material scris de echipa SO: Terminarea thread-urilor

**2.5 Thread-urile se împart în două categorii :**

* unificabile :
* permit unificarea cu alte threaduri care apelează pthread\_join
* resursele ocupate de thread nu sunt eliberate imediat după terminarea threadului, ci mai sunt păstrate până când un alt thread va executa pthread\_join (analog cu procesele zombie)
* threadurile sunt implicit unificabile
* detaşabile
* nu se poate executa un pthread\_join pe ele
* vor elibera resursele imediat ce se vor termina (analog cu ignorarea semnalului SIGCHLD în părinte atunci când se termină procesele copil)

**2.6 Atributele unui thread**

Atributele reprezintă o modalitate de specificare a unui comportament diferit de comportamentul implicit. Atunci când un fir de execuţie este creat cu pthread\_create se poate specifica un atribut pentru respectivul fir de execuţie. Atributele implicite sunt suficiente pentru marea majoritate a aplicaţiilor. Cu Funcţii pentru cleanup 7 ajutorul unui atribut se pot schimba:

* starea: unificabil sau detaşabil
* politica de alocare a procesorului pentru thread-ul respectiv (round robin, FIFO, sau system default)
* prioritatea (cele cu prioritate mai mare vor fi planificate, în medie, mai des)
* dimensiunea şi adresa de start a stivei

**3 WINNOWS**

**3.1 API**

Windows API este o interfață destinată programării aplicațiilor pentru sistemul de operare Microsoft Windows (API este acronimul din limba engleză pentru Application Programming Interface). Windows API este cunoscută, în general, cu numele de Win32 API, însă denumirea Windows API reflectă mai precis capacitățile și utilitatea sa, suportul atât pentru Windows 32 biți, cât și pentru Windows 64 biți.Microsoft Windows SDK ( en. Software Development Kit) conține documentația și unelte necesare programatorilor pentru a realiza aplicații folosind Windows API.Prin Win32 API programatorul are acces direct la o mare parte a funcțiilor de nivel de bază (en. low-level) ale sistemului de operare, putând crea aplicații într-un mod foarte flexibil.Windows API conține o ofertă de servicii pentru toate aplicațiile bazate pe ferestre grafice (en. windows). Această interfață de programare permite utilizatorilor să realizeze o interfață grafică propriilor aplicații, să acceseze sistemul computerului, memoria acestuia, dispozitivele (fie de intrare, fie de ieșire), să implementeze sunete, imagini, video sau funcțiuni de rețea (respectiv Internet) în aceste aplicații. Programarea cu Windows API înseamnă primirea, interpretarea, trimiterea de „mesaje” către „ferestre”, sau „controale” (obiecte controlabile - ex. ToolBox, EditBox, Button, Text, CheckBox).Funcția principală (en. main - WINAPI WinMain) a unei aplicații Win32 conține o „buclă” (structură iterativă) (ex. while() în limbajul C++) în care sunt apelate funcțiile de preluare și traducere a mesajelor trimise de către utilizator (prin intermediul dispozitivelor de intrare, apoi al sistemului de operare, în cazul de față Windows). Mesajele sunt interpretate și trimise mai departe ferestrei active. Mesajele pot fi trimise atât de sistem, asemenea unor mesaje din "subconștient", dacă ar fi să facem o comparație cu creierul uman, cât și explicit, "conștient", prin funcția SendMessage(). Fiecărei ferestre i se asociază o procedură responsabilă cu interpretarea mesajelor primite. Spre exemplu, dacă unei ferestre i se trimite un mesaj de „distrugere”, aceasta dispare. Dacă unui control de tip CheckBox i se trimite mesajul de validare, în dreptul său apare binecunoscutul marcaj de validare.

**3.2 Creearea unui Proces**

 Un proces este creat atunci cand un alt proces apeleaza metoda *CreateProcess* din biblioteca kernel32.dll.

Crearea procesului se face in urmatorii pasi :

* Fisierul executabil este analizat.Daca este un fisier executabil Windows pe 16 biti sau MS-DOS atunci este creat un mediu special de executie pentru acesta;daca fisierul este un executabil Windows pe 32 de biti se verifica registrul pentru a vedea daca are vreo cerinta speciala.Operatiile de la acest pas se fac in modul utilizator.
* Se creeaza un obiect proces gol cu apelul de sistem NtCreateProcess si se adauga in spatiul de nume al managerului de obiecte. Mai sunt create un obiect nucleu si unul executiv. Managerul de proiecte creeaza un bloc de control al procesului pentru acest obiect si il initializeaza cu Idul procesului si cu alte campuri.Este creat un obiect sectiune pentru a urmari spatiul de adrese al procesului.
* kernel32.dll preia controlul,efectueaza apelul de sistem *NtCreateThread* pentru a crea un fir de executie initial.Se creeaza stivele utilizator si nucleu.Dimensiunea stivei este data de antetul fisierului executabil.
* kernel32.dll trimite subsistemului de mediu Win32 referintele catre proces si catre firul de executie care sunt adaugate in niste tabele.
* Este apelata o procedura de sistem care termina initializarea iar firul de executie incepe sa ruleze.

procedura apelata la executie seteaza prioritatea firului de executie.Se repeta pasul pentru toate firele de executie.Incepe rualrea codului principal al procesului.

**3.3 Crearea firelor de executie**

Crearea firelor de execuţie Pentru a lansa un nou fir de execuţie există funcţiile CreateThread i CreateRemoteThread (a doua fiind folosită pentru a crea un fir de execuţie în cadrul altui proces decât cel curent).

HANDLE CreateThread (

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

SIZE\_T dwStackSize,

LPTHREAD\_START\_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId

);

dwStackSize reprezintă mărimea iniţială a stivei, în bytes. Sistemul rotunjete această valoare la cel mai apropiat multiplu de dimensiunea unei pagini. Dacă parametrul este 0, noul thread va folosi mărimea implicită. lpStartAddress este un pointer la funcţia ce trebuie executată de către thread. Această funcţie are următorul prototip:

DWORD WINAPI ThreadProc(

LPVOID lpParameter );

unde lpParameter reprezintă datele care sunt pasate firului de execuţie la execuţie. La fel ca pe Linux, se poate transmite un pointer la o structură, care conţine toţi parametrii necesari. Rezultatul întors poate fi obţinut de un alt thread folosind funcţia GetExitCodeThread.

Un mic exemplu :

HANDLE hthread;

hthread = CreateThread(NULL, 0, ThreadFunc, &dwThreadParam, 0, &dwThreadId);

La crearea unui nou fir de execuţie parametrii cei mai importanţi sunt funcţia pe care acesta o va executa şi parametrul care este pasat acesteia

**3.3.1 Aşteptarea firelor de execuţie**

Pe Windows, se poate aştepta terminarea unui fir de execuţie folosind aceeaşi funcţie ca pentru aşteptarea oricărui obiect de sincronizare:

DWORD WINAPI WaitForSingleObject(

HANDLE hHandle,

DWORD dwMilliseconds

 );

**3.3.2 Terminarea firelor de execuţie**

 Un fir de execuţie se termină în unul din următoarele cazuri :

* el însui apelează funcţia ExitThread
* funcţia asociată firului de execuţie execută un return.
* sau întregul proces se termină ca urmare a unui apel ExitProcess sau TerminateProcess.

**3.3.3 Suspend, Resume**

DWORD SuspendThread(HANDLE hThread);

DWORD ResumeThread(HANDLE hThread);

Prin intermediul acestor două funcţii un fir de execuţie poate suspenda/relua execuţia unui alt fir de execuţie. Un fir de execuţie suspendat nu mai este planificat pentru a obţine timp pe procesor. Cele doua funcţii manipulează un contor de suspendare (prin incrementare, respectiv decrementare - în limitele 0 - MAXIMUM\_SUSPEND\_COUNT). În cazul în care contorul de suspendare este mai mare strict decât 0, firul de execuţie este suspendat. Un fir de execuţie poate fi creat în starea suspendat folosind flag-ul CREATE\_SUSPENDED.

Aceste funcţii nu pot fi folosite pentru sincronizare (pentru ca nu controlează punctul în care firul de execuţie îi va suspenda execuţia), dar sunt utile pentru programe de debug.

**3.4 Fibre de execuţie**

Windows pune la dispoziţie şi o implementare de user-space threads, numite fibre. Kernel-ul planifică un singur KLT asociat cu un set de fibre, iar fibrele colaborează pentru a partaja timpul de procesor oferit acestuia. Deşi viteza de execuţie este mai bună (pentru context-switch, nu mai este necesară interacţiunea cu kernel-ul), programele scrise folosind fibre pot deveni complexe. Mai multe informaţii puteţi găsi la secţiunea suplimentară dedicată.

**3.5 Concluzii**

Pentru dezvoltarea de aplicatii distribuite in reteaua Internet, sistemul de operare Linux ofera programatorilor o interfata API Internet, cu sistemul de comunicatie, dupa modelul Berkeley Socket, concept introdus pentru prima data in anul 1981, odata cu versiunea BSD 4.1 UNIX pe de alta parte Windows API conține documentația și unelte necesare programatorilor pentru a realiza aplicații folosind Windows API.Prin Win32 API programatorul are acces direct la o mare parte a funcțiilor de nivel de bază (en. low-level) ale sistemului de operare, putând crea aplicații într-un mod foarte flexibil.Windows API conține o ofertă de servicii pentru toate aplicațiile bazate pe ferestre grafice (en. windows).

Creearea unui proces În Linux singurul mod prin care se poate creea un nou proces este prin apelul de sistem fork. Efectul acestui apel este creearea unui nou proces, copie a procesului care a apelat fork. Difera doar PID-ul proceselor, noul proces primind un alt PID de la sistemul de operare, in Window un proces este creat atunci cand un alt proces apeleaza metoda *CreateProcess* din biblioteca kernel32.dll.s

**Bibliografie:**

**Andrei ClubCisco**