Curs 1:

Din ce este compus un sistem de calcul modern?

Un sistem de calcul modern este compus din:

- Unul sau mai multe procesoare
- Memoria principala
- Unitati de disc
- Dispozitive de intrare/iesire (exemple?)

Care este rolul sistemului de operare?

Gestiunea tuturor acestor componente necesita un anumit nivel software – sistemul de operare.

Care sunt cele patru componente de bază ale unui sistem de calcul?

Exista patru componente de baza ale unui sistem de calcul:

- Componenta hardware
- Programele de aplicatie
- Sistemul de operare
- Utilizatorii

Cum acţionează sistemul de operare faţă de celelalte componente?

Sistemul de operare poate fi considerat ca un liant al celorlalte trei componente.

Cum extinde sistemul de operare maşina de bază?

Ofera un mediu in care cele trei componente de baza (hardware, software, date sistem) pot fi utilizate corect;

Cum sunt vizibile detaliile de funcţionare ale dispozitivelor?

Ascunde toate detaliile de functionare ale dispozitivelor hardware, oferind o interfata uniforma pentru utilizarea acestora (prin interfata apelurilor sistem).

Cum se manifestă sistemul de operare ca manager de resurse?

Permite ca mai multe programe sa ruleze in acelasi timp

Care sunt principalele resurse vizate?

Gestioneaza si protejeaza memoria, dispozitivele de intrare / iesire, precum si alte resurse

Ce puteţi spune despre multiplexarea resurselor? (cele două modele de bază)

Resursele multiplexate (partajate) sunt utilizate (incluse) in doua moduri diferite:

- In timp
- In spatiu

Cum se manifestă sistemul de operare ca program de control?

Sistemul de operare ca un program de control:
Este responsabil cu controlul executiei programelor pentru a preveni utilizarea incorecta sau ilegala a resurselor oferite de catre sistemul de calcul

CURS 2:

Care au fost evenimentele care au dus la un progres remarcabil în acest domeniu?

- (1945–55) Tuburi vidate
- (1955–65) Transistori si sisteme cu prelucrare in loturi
- (1965–1980) IC si Multiprogramare
- (1980–Prezent) Calculatoare personale

Pentru prima generaţie de calculatoare, identificaţi caracteristicile acestora:

- “hardware-ul” implicat
- Viteza de calcul
- Cost de operare
- Mijloace de operare
- Suport prin limbaje de programare
- Sisteme de operare

Monitorul rezident Este o componenta destinata utilizarii cartelelor de control, oferind suportul pentru succesiunea automata a lucrarilor; Primul “sistem de operare” real; Monitorul este un program de dimensiuni mici, pastrat permanent in memorie; Scopul principal este de a transfera controlul catre programe si de a prelua controlul de la acestea, indata ce sarcina acestora s-a incheiat.

Monitorul rezident are trei componente de baza: Incarcatorul, interpretorul de cartele de control, driverele de dispozitiv.

Procesarea off-line s-a bazat pe dezvoltarea benzilor magnetice. Programele sunt citite de la o unitate de banda, iar rezultatele sunt depozitate pe alte unitati de banda. Operarea ulterioara a benzilor magnetice a fost realizata “la distanta” pe masini dedicate pentru aceste operatii simple.

Spooling

- Revenirea la operarea on-line;
- Simultaneous Peripheral Operation On-Line se bazeaza pe unitatie de disc, utilizate pentru depozitarea “permanenta”;
- Spooling-ul este posibil datorita dezvoltarii dispozitivelor de stocare cu acces aleator.

Multiprogramarea

- Se bazeaza pe ideea de partitionare a memoriei principale: aceasta poate mentine mai multe lucrari simultan;
- Multiprogramarea imbunatateste tehnica spooling;
- Cere dezvoltareaunor tehnologii noi de protectie la nivel hardware sau software;
- Odata cu multiprogramarea, procesorul are posibilitateade a alege urmatorul proces de executat. Planificarea reala a proceselor devine astfel posibila.

Sistemele monolitice – structura de baza:

- Un program principal care invoca procedurile de servicii necesare.
- Un set de proceduri de servicii care duc la indeplinire apelurile sistem.
- Un set de utilitare care vin in ajutorul procedurilor de servicii.

Doua dintre sistemele de operare des utilizate au fost caracterizate printr-o structura monolitica:

- ***Sistemul MS-DOS;*** Este un SO cu o structura “slaba”, fara o delimitare clara a interfetelor sau a nivelelor de functionalitate. MS-DOS ofera o abordare simplificata a dispozitivelor I/O: fiecare aplicatie poate accesa direct orice dispozitiv I/O.

Masinile virtuale au fost dezvoltate odata cu primele masini cu partajarea timpului:

- O masina virtuala trebuie sa ofere o extensie a masinii, printr-o interfata convenabila;
- O masina virtuala trebuie sa ofere serviciile de baza necesare multiprogramarii.
- Totodata, masina virtuala trebuie sa realizeze o separare clara a acestor doua functii.

VM/370 - masina virtuala
MINIX3 - micronucleu
THE - sistem stratificat
CP/M - monolitic

CURS 3:

Multiprogramarea si partajarea timpului ofera posibilitatea ca procesele multiple si/sau utilizatorii sa existe simultan in sistem, ridicand numeroase probleme de protectie.

Modul dual de operare este folosit ca un mecanism de protectie prin specificarea unui set de instructiuni procesor care pot fi executate doar cand acest mod de operare este activ (instructiuni privilegiate). Aceste instructiuni vor fi destinate numai sistemului de operare. Modul dual de operare a fost implementat initial la nivel software, pentru a limita accesul catre anumite servicii, destinate doar sistemului de operare. Mai tarziu, modul dual de operare a fost oferit la nivel hardware, printr-un bit specializat (numit bit de mod), controlat doar de catre sistemul de operare.

I/O protection: Sistemul de calcul defineste operatiile de intrare/iesire ca fiind privilegiate, accesibile doar in modul monitor.
Deoarece intreruperile sunt mijlocul preferat de rezolvare a operatiilor de intrare/iesire, mecanismul intreruperilor ar putea oferi posibilitatea de a castiga controlul asupra sistemului de catre procese utilizator. O intrerupere este rezolvata cu ajutorul unui vector de intreruperi.

Protecţia memoriei: Pentru a avea un sistem de protectie a memoriei efectiv, este necesara utilizarea unor componente hardware specifice, de exemplu bazate pe manipularea registrilor baza si limita. Protectia este realizata la niveul unitatii de procesare, prin compararea adreselor generate cu valorile depozitate in acesti registri.

Protectia procesorului: Protectia procesorului necesita o abordare diferita. Aceasta se datoreaza faptului ca, in timpul executiei SO, procesorul este oferit periodic proceselor utilizator. Solutia este inspirata de mecanismele time-sharing, si se bazeaza pe un cronometru. Acesta este fixat pentru o perioada fixa sau variabila de timp, predefinita.

‘recapitulare’

Caracteristici procesor:

- Setul de instructiuni
- ALU Arithmetic/Logic Unit
- Contorul program
- Pointerul de stiva
- Registrii de uz general
- PSW – cuvantul de stare al programului
- Pipeline

Dispozitive I/O: Procesarea intreruperilor presupune preluarea intreruperii, executarea handlerului intreruperii, returnarea controlului catre programul utilizator.

Tipuri de sisteme de operare:

- Sisteme de operare Mainframe
- Sisteme de operare Server
- Sisteme de operare Multiprocesor
- Sisteme de operare de tip PC
- Sisteme de operare “Handheld”
- Sisteme de operare “Embedded”
- Sisteme de operare in timp real
- Sisteme de operare “Smart card”

Concepte ale sistemelor de operare:

- Procese
- Spatiul de adrese
- Fisiere
- Intrare/Iesire
- Protectie
- Interpretor de comenzi
- Ontogenie repeta filogenia: - Memorie de dimensiuni mari
  - Dispozitive (hw) de protectie
  - Discuri
  - Memorie virtuala

Apelurile sistem ofera interfata dintre procese si sistemul de operare.De regula sunt oferite ca instructiuni intr-un limbaj de asamblare, sau chiar ca instructiuni in limbaje de nivel inalt. Intreaga functionare a sistemului de operare se bazeaza pe apelurile sistem: de regula procesele cer servicii de la Sistemul de Operare prin interfata apelurilor sistem.

Un proces este un program in executie. Pentru fiecare proces exista un spatiu de adrese asociat. Executia proceselor se realizeaza intr-o maniera secventiala, instructiune dupa instructiune.

Managementul proceselor este realizat prin apeluri sistem specifice, principalul program care le utilizeaza fiind interpretorul de comenzi (shell-ul).

Sarcinile unui sistem de operare:

- Crearea si “stergerea” proceselor;
- Suspendarea si continuarea proceselor;
- Sincronizarea proceselor prin mecanisme specifice;
- Comunicarea intre procese prin mecanisme specifice;
- Rezovlarea situatiilor de impas prin mecanisme specifice.

Apeluri sistem tipice pentru gestiunea proceselor

- Procesele UNIX sunt bazate pe mecanismul fork()...exec().
- In cazul Win32 se utilizeaza apelul CreateProcess().
- Terminarea proceselor este realizata prin apelulexit() – UNIX sau ExitProcess() – WIN32.
- Asteptarea terminarii proceselor se realizeaza prin apelul wait() – UNIX, WaitForSingleObject() – WIN32.
- Apelul kill() – UNIX ofera atat un mecanism rudimentar de comunicare intre procese cat si o cale de terminare fortata a unui proces.

Fisierul este unitatea logica de depozitare a informatiei intr-un sistem de calcul. Un fisier este, in cea mai simpla definitie, o simpla secventa de octeti.

Directoarele au fost dezvoltate pentru a oferi un mijloc simplu de grupare si organizare a fisierelor.

Gestiunea fisierelor (apeluri sistem)

- open() – UNIX, CreateFile() – Win32, pentru a deschide un fisier.
- Crearea fisierelor, prin open() sau creat() – UNIX, CreateFile() – Win32.
- Inchiderea unui fisier prin close() sau CloseHandle().
- Citirea/scrierea datelor, prin read()/write() sau ReadFile()/WriteFile().
- Pozitia pointerului intern al fisierului, controlata prin lseek() sau SetFilePointer().
- Informatii despre fisiere se pot obtine prin …stat() sau GetFileAttributesEx().

Gestiunea directoarelor (apeluri sistem)

- Crearea si stergerea directoarelor, prin mkdir()/rmdir(), sau CreateDirectory()/RemoveDirectory().
- Schimbarea directorului curent (director de lucru) prin chdir() sau SetCurrentDirectory().
- Stergerea unei intrari de director, prin unlink() sau DeleteFile().

Memoria principala este principalul “depozit” al programelor in executie. Programele in executie sunt totdeauna depozitate in memoria principala a sistemului.

Sarcinile sistemului de operare (relativ la memoria principala):

- Urmarirea utilizarii memoriei;
- Realizarea de decizii asupra proceselor care urmeaza sa fie depozitate in memorie;
- Alocarea si dealocarea memoriei.

CURS 4-5:

Cum poate fi definit un proces?
Un proces este un program in executie, impreuna cu valorile asociate acestuia;

- Un sistem poate pastra la un moment dat mai multe procese, incluzand si sistemul de operare;
- Sistemul de operare ofera mecanismele necesare pentru operatia de comutare de proces, ascunzand executia secventiala a proceselor in spatele iluziei de paralelism;

Care este mecanismul de realizare a multiprogramării?
Multiprogramarea este bazata pe operatia de comutare a proceselor; Prin aceasta operatie, procesorul poate fi oferit alternativ fiecarui proces;

Ce presupune operaţia de comutare de context?
In cazul operatiei de comutare de proces, vor fi avute in vedere limitarile hardware existente:

- De regula exista un singur set de registrii care sunt utilizati direct pentru executie, chiar daca procesorul poate oferi mai multe seturi de registri.

Sistemul de operare trebuie sa foloseasca aceasta caracteristica pentru a rezolva toate operatiile legate de comutarea de context (salvare si restaurare de context);

Care sunt factorii care pot influenţa realizarea acestei operaţii?
Sistemul de operare nu va face niciun fel de presupuneri legate de timpii de executie a proceselor.

Puteţi identifica evenimentele care duc la crearea unui proces?
Evenimente care pot determina crearea proceselor:

- Initializarea sistemului.
- Executia unui apel sistem pentru crearea proceselor de catre un proces.
- O cerere utilizator pentru crearea unui proces nou.
- Initierea unei prelucrari in loturi.

Care sunt evenimentele care sunt iniţiate de un alt proces? Dar cele iniţiate din exterior?
La initializarea sistemului, sistemul de operare va creea procesele necesare pentru utilizarea corecta a sistemului

- Ex. Procese care suporta interactiunea utilizat; procese destinate rezolvarii unor sarcini specifice.

In cazul sistemelor de operare moderne, procesele noi sunt capabile sa creeze alte procese. Exista diferite motive pentru un asemenea comportament:

- ▪ Oferirea de servicii specializate;
- ▪ Oferirea sprijinului pentru rezolvarea de probleme;

        ▪ Oferirea sprijinului pentru executia proceselor inrudite; etc.

Pentru sistemele interactive, cererile utilizator pot fi utilizate pentru crearea de procese noi.

- Interactiunea cu utilizatorul este posibila prin utilizarea unui interpretor de comenzi; procesele nou create sunt (de regula) instante ale unor comenzi sistem sau programe utilizator.

Pentru sistemele cu prelucrare in loturi, crearea proceselor se poate realiza in timpul transmiterii unei lucrari catre sistem, sau ca urmare a prelucrarii unei cartele de control.

- Sistemul de operare trebuie sa decida asupra creerii unei lucrari noi doar atunci cand resursele disponibile sunt suficiente pentru aceasta operatie.

Care sunt mecanismele legate de implementarea operaţiei de creare a proceselor?

Mecanismul de creare a proceselor este urmatorul:

- Un proces existent emite un apel sistem pentru a creea un proces nou (ex. Apelul sistem fork()). Anumite sisteme de operare limiteaza aceasta posibilitate la anumtie categorii de procese.
- Sistemul de operare poate fi instiintat, in acelasi timp, asupra necesitatii de a incarca (sau executa) un alt program in spatiul nou creat (ex. Printr-un apel sistem de tip exec…()).
- Alte sisteme de operare, cum ar fi sisteme de tip Windows, ofera o functie unica (si in acelasi timp complexa) pentru a rezolva toate cerintele operatiei de creare a proceselor.

Puteţi identifica evenimentele care duc la terminarea unui proces?

Evenimente care pot determina terminarea proceselor:

- Terminare normala (voluntar).
- Terminare printr-o eroare (voluntar).
- Eroare fatala (involuntar).
- Terminare datorata unui alt proces (involuntar).

Care sunt evenimentele care sunt iniţiate de către procesul însuşi? Dar cele iniţiate din exterior?

- Majoritatea proceselor dintr-un sistem de operare sunt caracterizate printr-o terminare normala, odata emis un apel sistem specific; Erorile fatale reprezinta evenimente speciale in timpul executiei proceselor (ex. Ca urmare a utilizarii unui nume de fisier invalid).
    - ★ Procesele nu-si pot continua executia, ca urmare a erorilor fatale;
- Situatiile de exceptie (ex. Impartire la zero, acces ilegal la memorie, operatii invalide cu pointeri) ofera un alt tip de motiv pentru terminarea proceselor.
    - ★ Exceptiile pot fi interceptate si manipulate. Pentru acest tip de evenimente, terminarea proceselor poate fi evitata sau amanata;
- Terminarea fortata poata sa apara doar ca urmare a cererii unui alt proces, autorizat pentru acest tip de operatie.

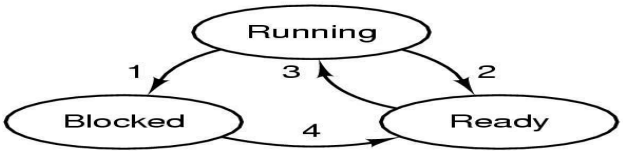
Care sunt legăturile dintre procese? Există o legătură părinte-copil? În care sisteme de operare?
Poate fi implementată o ierarhizare a proceselor? Dacă da, cum?
Diferitele sisteme de operare ofera mecanisme diferite pentru a pune in evidenta relatiile dintre procese.

- Pentru sistemele UNIX este implementata o relatie de tip parinte-copil.
- Toate procesele sunt descendetul unui unic proces, init, creat odata cu initializarea sistemului (traditional, valoarea PID – Process ID – este 1).
- Pentru sistemele de tip Windows, “creatorul” unui proces este inzestrat cu un token special prin care poate fi utilizat pentru a controla procesul “creat”. Cu toate acestea, in filozofia Windows, toate procesele sunt egale.

Puteţi identifica principalele evenimente legate de execuţia proceselor?

- Starile unui proces reflecta principalele momente in cursul executiei acestora. Primele sisteme folosesc un model simplu, cu 3 stari:
    1. “in curs de executie”
    2. “pregatit”
    3. “blocat”.
- Procesele sunt blocate atunci cand asteapta aparitia unui eveniment extern (de regula, o operatie I/O, dar si activitatea unui planificator sau altele)
- Fiecare Sistem de operare implementeaza o schema proprie de tranzitie intre stari, in conformitate cu politicile sistemului in ceea ce priveste gestiunea proceselor.

Cum se poate realiza tranziţia între aceste stări?



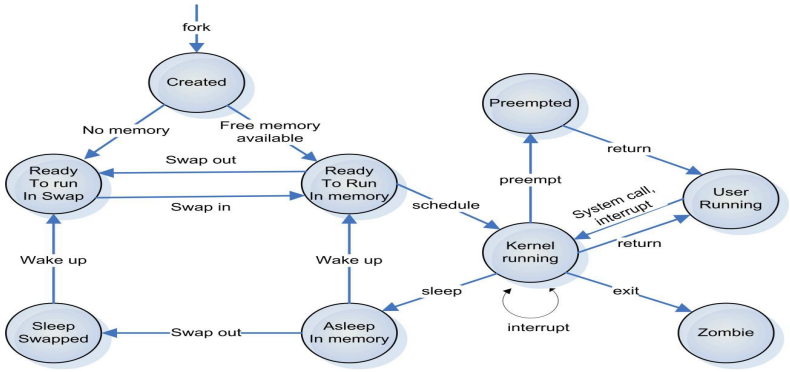
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Un proces se poate gasi in stare “running” (in curs de executie), “blocked” (blocat) sau “ready” (pregatit). Tranzitiile dintre aceste stari, pentru modelul simplu cu trei stari, sunt evidentiate.

Cum credeţi că influenţează mecanismele de memorie virtuală stările din cursul execuţiei proceselor?

- Modelul clasic ofera o viziune de ansamblu asupra executiei proceselor.
- Ca urmare a introducerii tehnicilor de memorie virtuala, modelul cu 3 stari poate fi modificat pentru a oferi stari suplimentare, asociate mecanismelor de swapping.
    - ★ Un proces blocat (in memorie) poate fi transferat pe disc, ca proces suspendat. Activitatea unui proces suspendat poate fi reluata prin readucerea acestuia in memorie ca proces “pregatit pentru executie”.

Procesele suspendate nu pot fi incarcate in memorie intr-o alta stare decat daca fac obiectul unei activitati de planificare.



Cum păstrează sistemul de operare informaţii legate de procese? Care sunt aceste informaţii?Informatiile sunt salvate in memorie intr-o tabela specializata de procese.

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Cum credeţi că se poate defini un fir de execuţie faţă de un proces?
Oferiţi definiţii posibile acestei noţiuni.

- Procesele clasice sunt caracterizate prin executii unice, urmand un singur fir (sau directie) de executie. Acest singur fir ofera un model simplu, secvential.
- Firele de executie ofera o extensie a modelului clasic al proceselor, prin implementarea de executii paralele care pot fi realizate in spatiul unui singur proces, aceste fire avand de asemenea un anumit grad de independenta.
- Firele de executie folosesc o cantitate limitata de informatii, definite in contextul procesului “gazda”. Aceasta limitare se refera doar la informatii care sunt strict legate de executia curenta.

Sunt firele de execuţie orientate către calcul sau către gestiunea resurselor?
Firele de executie ruleaza in contextul unui anumit proces, mostenind o parte dintre proprietatile acestuia. Uneori este folosit termenul de “proces usor” pentru a face referire la fire de executie. Se presupune ca firele de executie sunt orientate mai degraba catre executii (calcul), in locul gestiunii resurselor existente (si partajate) in contextul procesului curent.

Care sunt diferenţele dintre execuţiile orientate către calcul şi cele orientate spre gestiune de resurse?
Se presupune ca firele de executie sunt orientate mai degraba catre executii (calcul), in locul gestiunii resurselor existente (si partajate) in contextul procesului curent.

Care credeţi că sunt informaţiile necesare pentru implementarea firelor de control a execuţiei?

INFORMATII PARTAJATE DE TOATE FIRELE DE EXECUTIE DINTR-UN PROCES

- Spatiul de adrese
- Variabile globale
- Fisiere deschise
- Procese copil
- Alarme in asteptare
- Semnale si handlere asociate
- Informatii de “contabilitate”

INFORMATII PRIVATE PENTRU FIECARE FIR

- Contorul program
- Stiva
- Registrii
- Starile.

Sunt procesele independente sau interdependente?

Sunt firele de control a execuţiei independente sau interdependente?

- Exista un inalt grad de interdependenta intre firele de executie. Aceasta se datoreaza faptului ca firele de executie ruleaza in acelasi spatiu proces, si au un acces uniform catre resursele disponibile procesului “gazda”.
- Fiecare fir de executie poate interactiona cu executia unui alt fir de executie.
- Nu exista niciun mecanism de protectie la nivel de fir. Acest tip de protectie nu este necesar, avand in vedere caracteristicile principale ale firelor:
    - ★ Oferirea de executii paralele pentru a rezolva (colaborativ) problemele in acelasi spatiu cu procesul gazda.

Cum credeţi că este implementat multithreadingul?

Cum influenţează interdependenţa firelor de control a execuţiei acest mecanism?

- Spre deosebire de multi-tasking, multi-threadingul se bazeaza pe presupunerea ca diferitele fire ale aceluiasi proces pot partaja diferite resurse logice care sunt utilizate de catre proces.
    - ★ Multi-threadingul nu implica neaparat suportul pentru multitasking din partea sistemului de operare.
- Multi-threadingul este oferit intr-o maniera similara cu multi-programarea: alternarea executiilor diferitelor fire printr-o permanenta comutare intre firele care se executa.
    - ★ Mecanismul de comutare poate fi coordonat de catre sistemul de operare insusi, sau poate fi “comandat” de catre un proces, functie de modelul pentru fire de executie utilizat.

Puteţi identifica stările firelor de control, prin comparaţie cu procesele?

Pentru firele de executie poate fi imaginat un model similar cu cel pentru procese, cu stari de baza si tranzitii similare:

- Fire “pregatite pentru executie”, orice fir care este capabil sa fie executat, indata ce firul este planificat pentru executie.

- Fir “in executie”, este singurul fir activ, firul care controleaza procesul curent.
- Fire “blocate”, sunt fire care asteapta anumite evenimente. Totusi, un fir poate fi blocat si la cererea unui alt fir!
- Fire “incheiate”, sunt fire care si-au terminat activitatea, si asteapta raportarea rezultatelor executie (prin mecanismul join).

Există mai multe modele de implementare a firelor de control a execuţiei.

O abordare posibila in cazul sistemelor de operare simple, cum ar fi SO mono-utilizator, pentru sisteme de operare care nu ofera suport pentru executia firelor sau pentru orice alt sistem de operare. Executia firelor este realizata doar in spatiul utilizator: SO nu are niciun fel de informatie despre aceste executii. Intregul management este rezolvat la nivel de proces, printr-un subsistem specializat de management al firelor. Acest management este bazat pe o tabela de fire, insotita de decizii similare deciziilor de planificare.

Al doilea model presupune un suport deosebit din partea sistemului de operare.

De aceasta data activitatile de management ale firelor sunt “pasate” sistemului de operare. SO va oferi apeluri sistem specifice pentru gestiunea firelor, si va mentine informatii suplimentare pentru acest scop. Acum firele blocate nu mai blocheaza intreaga aplicatie SO este capabil sa planifice de data aceasta alte fire ale aceleiasi aplicatii, daca este cazul. Crearea si stergerea firelor este insa costisitoare, fiind similare cu operatiile pentru procese. Pentru a trece peste aceasta dificultate, SO poate mentine un set de socluri-fir, iar firele sterse vor fi doar marcate, insa pastrate in sistem pentru a grabi operatii viitoare de creare a firelor noi.

A treia abordare: abordarea hibrida

De regula SO nu se limiteaza la una dintre cele doua implementari precizate anterior. O abordare hibrida este utilizata pentru a combina avantajele firelor in spatiul utilizator cu managementul firelor in spatiul nucleu. In abordarea hibrida, firele sunt mapate in spatiul nucleu, astfel incat vor putea fi oferite mecanismele tipice pentru gestiunea firelor (si proceselor). Nucleul SO este insa interesat doar in aceste activitati de management, fara un interes deosebit pentru obiectele gestioante (care acum exista in spatiul utilizator).

Motivele folosirii firelor de executie

- Procesare “foreground” si “background”: o aplicatie ar putea folosi tehnicile multifir pentru a afisa un meniu, pentru a cauta un document, dar si pentru o activitate de update, realizate in acelasi timp.
- Procesari asincrone: activitatile asincrone ar putea fi implementate intr-o aplicatie folosind diferite fire de executie. De exemplu, o activitate de update, o activitate de auto-salvare a documentelor ar putea fi considerate activitati tipic asincrone.
- Imbunatatirea vitezei de executie: diferitele fire ar putea fi utilizate pentru a realiza suprapunerea operatiilor.

CURS 6:

Procese cooperante

★ Un proces este independent daca nu este afectat si nu poate afecta un alt proces (in executie).

★ Un proces care nu este independent este … cooperant. De regula, procesele cooperante partajeaza anumite resurse (e.g. fisiere PIPE, zone de memorie, altele) cu alte procese. Cooperarea este necesara pentru a:
- ▪ Partaja informatii.
- ▪ Obtine o viteza superioara de calcul.
- ▪ Oferi modularitate, eventual prin oferirea de sarcini elementare catre procese diferite.
- ▪ Oferi convenienta, pentru rezolvarea unui numar mare de sarcini simultan.

Obiectivul multiprogramarii: maximizarea utilizarii procesorului

- Sistemele cu partajarea timpului utilizeaza o frecventa comutare a resurselor de calcul intre diferitele procese;
- Interactiunea cu utilizatorul se bazeaza pe acest mecanism de comutare.

Pentru sistemele simple, mono-procesor, multiprogramarea si partajarea in timp sunt notiuni similare

- Procesorul este oferit alternativ proceselor printr-o comutare frecventa intre acestea.

Cozi de planificare Intr-un sistem, procesele se afla permanent in una dintre cozile acestuia:

- ***COADA LUCRARILOR: contine procesele acceptate in sistem;***
  - Aceasta coada ofera intreaga plaja de lucrariexistente.
- ***COADA PROCESELOR PREGATITE PENTRU EXECUTIE: contine … procesele pregatite pentruexecutie;***
  - Aceste procese se afla in memorie (daca este posibil, toate procesele pregatite se gasesc in aceasta coada).
  - BCP trebuie sa contina un camp specializatpentru a suporta aceasta coada. ▪ Planificarea procesorului este puternicbazatape aceasta coada.
- ***COZILE I/O & DE INTRERUPERI: contin procese blocate***
  - De asemenea, procesele blocatese pot regasi in coada proceselor carora le-a expirattimpul alocat.

Niveluri de planificare

- Procesele depozitate in diferitele cozi concureaza pentru acces la diferite resurse.
  - Intreaga existenta a unui proces este bazata pe o permanenta migrare intre diferitele cozi.
- Sistemul de operare, prin componentele sale specializate (planificatori) este responsabil cu alegerea proceselor care urmeaza sa acceseze aceste resurse.
- Aceasta alegere este bazata pe un algoritm de planificare, si trebuie sa asigure un bun echilibru intre diferitele procese existente:
  - Nivelul de multiprogramare al sistemului trebuie sa fie mentinut la un nivel cat mai constant posibil.

        - Acesta este reprezentat de numarul de procese care se afla simultan in memorie
- Sistemul trebuie sa suporte doua categorii majore de procese Procese marginite la operatii I/O
    - Procese care petrec majoritatea timpului pentru a rezolva operatii I/O in defavoarea calculului. Procese marginite la procesor
    - Procese care petrec majoritatea timpului pentru a realiza calcule in defavoarea operatiilor I/O.

Categorii de algoritmi de planificare

- Batch – simpli, dar efectivi
- Interactivi
- Real-time – complecsi, bazat pe politica sistemului.

Categorii de procese. Cele doua categorii majore considerate pentruactivitatile de planificare:

- Procesemarginite (limitate) la operatiiI/O;
- Procesemarginite (limitate) la procesor.

Momentele planificarii. Deciziile de planificare trebuie sa fie luate din motive diferite, la momente diferite in cursul vietii unui proces. Aceste momente pot include.

- Creareaunui nou proces;
- Terminarea unuiproces;
- O intrerupere pentruo cerere I/O;
- Blocarea/suspendareaunui proces;
- Aparitiaunor alte intreruperi(trap, e.g. timer trap).

Algoritminon-preemptivi.

- Algoritmi utilizati pentru situatii in care activitatea proceselor nu poate fi intrerupta.
    - Sistemele cu prelucrare in loturi (batch) prefera acesti algoritmi.
    - Eventual sunt folositi algoritmi preemptivi cu limite largi de timp.

Algoritmi preemptivi.

- In acest caz procesele sunt alese pentru executie pentru o perioada limitata de timp.
    - Sistemele interactive prefera acesti algoritmi.
    - Algoritmii non-preemptivi nu sunt potriviti pentru comportament interactiv.

Cerinte planificare procese:

- ***Corectitudine:*** procesele comparabile vor fi tratate in acelasi mod. Acestea vor avea aceeasi sansa de a ajunge la resurse.
- ***Echilibrul:*** activitatile de planificare vor oferi un nivel inalt de utilizare a majoritatii componentelor sistemului. Prin aceasta cerinta se poate atinge un nivel inalt de productivitate. De exemplu, s-ar putea evita “preferarea” proceselor limitate la procesor in detrimentul celor limitate la operatii I/O.

- ***Respectarea politicilor:*** algoritmii de planificare se vor adapta la politicile locale ale sistemului, impuse de cerintele specifice ale sistemului de calcul.

*Pentru algoritmii de planificare **'batch'**, se va urmari evolutia unor parametrii, incluzand:*

- ***Iesirea:*** numarul de lucrari procesate in unitate ade timp (de exemplu, o ora).
- ***Timpul de intoarcere:*** pentru un proces, acesta este diferenta dintre momentul la care lucrarea isi termina executia si cel la care lucrarea a fost oferita sistemului. ▪ La nivel de algoritm se considera valoarea medie.
- ***Utilizarea procesorului:*** cerinta impusa de costul inalt al sistemelor de calcul de acest tip

*Pentru **sistemele interactive**, se urmareste evolutia unor parametrii, incluzand:*

- ***Timpul de raspuns:*** diferenta dintre momentul transmiterii si cel al obtinerii rezultatelor.
- ***proportionalitate:*** timpul de asteptare trebuie mentinut in limite rezonabile, acceptabile de catre utilizatorii finali.
- ***Timpul de asteptare:*** acesta reprezinta suma tuturor perioadelor de inactivitate ale unui proces. ▪ Se va calcula la nivel de algoritm timpul mediu de asteptare.

In timpul evaluarii diferitilor algoritmi de planificare, vor fi urmariti diferiti parametrii, incluzand:

- 1. Timpul mediu de raspuns (de intoarcere);
- 2. Timpul mediu de asteptare;
- 3. Momentul sosirii; momentul la care un proces este acceptat in sistem.
- 4. “Burst” ; durata unui ciclu de executie tipic.
- 5. Operatii de comutare context; numarul total de operatii de comutare context necesare.

FCFS (First Come First Served), sau FIFO Cel mai simplu algoritm de planificare.

- Procesele sunt alocate procesorului in ordinea in care sosesc.
- Toate procesele utilizeaza aceeasi coada de asteptare.
- Cand un proces este intrerupt (temporar), activitatea sa este reluata la capatul cozii de asteptare.
- Acesta este un algoritm non-preemptiv.

Shortest Job First (SJF)

- De asemenea un algoritm non-preemptiv, bazat pe durata (estimata) de executie a procesului.
- La fiecare pas este ales procesul cu cea mai scurta durata estimata de executie.
- Cand mai multe procese au aceeasi durata estimata de executie, FIFO (FCFS) este utilizat pentru decizia finala.
- Algoritmul SJF poate fi utilizat mai degraba pentru planificarea pe termen lung.
- Acest algoritm ofera valoarea optima pentru timpul mediu de asteptare. Din nefericire, este dificil de obtinut o estimare corecta a timpului de executie.

Shortest Return Time Next (SRTN)

- Acest algoritm este similar cu SJF.
- De data aceasta este folosit timpul estimat pentru terminarea ciclului de executie curent.
- Deciziile planificatorului au loc acum si la sosirea unui nou proces in sistem, precum si la terminarea unui proces.
- La sosirea unui nou proces, procesul curent este evacuat daca durata estimata a noului proces este mai scurta decat a procesului curent; altfel procesul curent isi continua executia.
- Este un algoritm preemptiv (de regula, identificat ca SJF preemptiv). Este folosit si numele “shortest remaining time first”.

Round Robin (RR)

- Unul dintre cei mai vechi algoritmi de planificare pentru sisteme interactive (inca in functiune).
- Pentru fiecare proces este alocata o cuanta de timp: cea mai mare felie de timp permisa pentru executia unui ciclu al unui proces.
- Un proces a carui felie de timp a expirat este evacuat din memrie si pus la capatul cozii de procese pregatite pentru executie.
- Valoarea aleasa pentru cuanta trebuie sa fie cat mai mica posibil, astfel incat timpul “pierdut” de catre procesor cu operatii de comutare de context sa fie cat mai mic posibil.
- Performanta intregului algoritm depinde puternic de valoarea cuantei. ▪ Pentru valori foarte mari ale cuantei, algoritmul este similar cu FCFS. ▪ Pentru valori foarte mici, abordareaRR este de “partajare a procesului”
- Valoarea cuantei va acoperi, pe cat posibil, cca 80% din timpii estimati de executie. In acest mod, cca 20% dintre procese vor avea nevoie de operatii de comutare de context.
- Timpul de raspuns este, de asemenea, afectat de aceasta decizie: valori mai bune sunt obtinute cand majoritatea proceselor se descurca cu o singura cuanta de timp.

Lottery Scheduling

- Acest algoritm ofera fiecarui proces cate un bilet de loterie.
    - Fiecare decizie este realizata printr-o “extragere”, premiul constand in resursele procesorului (o felie de timp).
- Sunt permise activitati adiacente, cum ar fi:
    - Un proces poate detine mai multe bilete (prin aceasta isi poate mari sansele: util pentru implementarea unei scheme bazata pe prioritati).
    - Tranzactiile cu bilete sunt permise.
        - Un grup de procese client ar putea coopera pentru a creste sansele procesului server cu care acestea comunica, atunci cand o activitate critica urmeaza sa aiba loc.

CURS 8-9:

Impas

O multime de procese se afla intr-o situatie de impas daca fiecare proces din multime asteapta aparitia unui eveniment care poate fi declansat doar de un alt proces din multime.

Sunt necesare patru conditii pentru o situatie de impas:

1. Excluderea mutuala: fiecare resursa este fie asignata exact unui proces, fie este disponibila;
2. 'Hold and wait': daca un proces detine resurse, acesta este capabil sa ceara si alte resurse;
3. Non-preemptiunea: nicio resursa nu poate fi obtinuta fortat de la niciun proces;
4. Asteptarea circulara: se poate identifica un lant de procese, unde fiecare proces asteapta o resursa care apartine unui proces precedent in acest lant.

Pentru **modelarea unei situatii de impas**, se poate folosi un graf orientat. Procesele sunt reprezentate prin cercuri. Resursele sunt reprezentate prin patrate.

Strategii pentru rezolvarea situatiilor de impas:

- Ignorarea problemei.
- Detectie si recuperare. Situatia de impas poate sa apara, este detectata si apoi remediata.
- Evitare dinamica printr-o alocare atenta a resurselor.
- Preventie, printr-o negare structurala a uneia dintre cele patru conditii necesare.

Detectie si recuperare: Aceasta tehnica poate fi utilizata cand, in locul prevenirii situatiilor de impas, sistemul prefera sa detecteze aparitia unei situatii de impas si sa reactioneze doar in acest moment.

Un sistem in care nu se poate pune in evidenta un ciclu in graful de resurse, este **liber de impas.**

Algoritmul de detectie:

1. Se cauta un proces nemarcat Pi, astfel incat linia acestuia in matricea R sa fie mai mica sau egala cu valorile vectorului A (adica se pot satisface cerintele de alocare ale procesului).
2. Daca exista un asemenea proces, adauga linia i din matricea C la vectorul A, marcheaza procesul, si treci la pasul 1.
3. Daca nu exista un asemenea proces, STOP.

4. Dupa executia algoritmului, toate procesele care nu sunt marcate se gasesc intr-o situatie de impas. Toate procesele marcate sunt “libere de impas”.

Algoritmul Bancherului (datorat lui Dijkstra) este utilizat pentru a verifica daca prin satisfacerea cererilor unui proces (prin planificarea acestuia) este posibila atingerea unei stari nesigura. Cererea este refuzata intr-o astfel de situatie.

CURS 10-11:

Există trei cerinţe de bază pentru comunicarea între procese:

1. Stabilirea unui mecanism pentru transmiterea informaţiilor între procese.
2. Evidenţierea mijloacelor prin care se pot garanta activităţile proceselor
3. Oferirea unor mecanisme pentru asigurarea ordinii corecte de execuţie a proceselor.

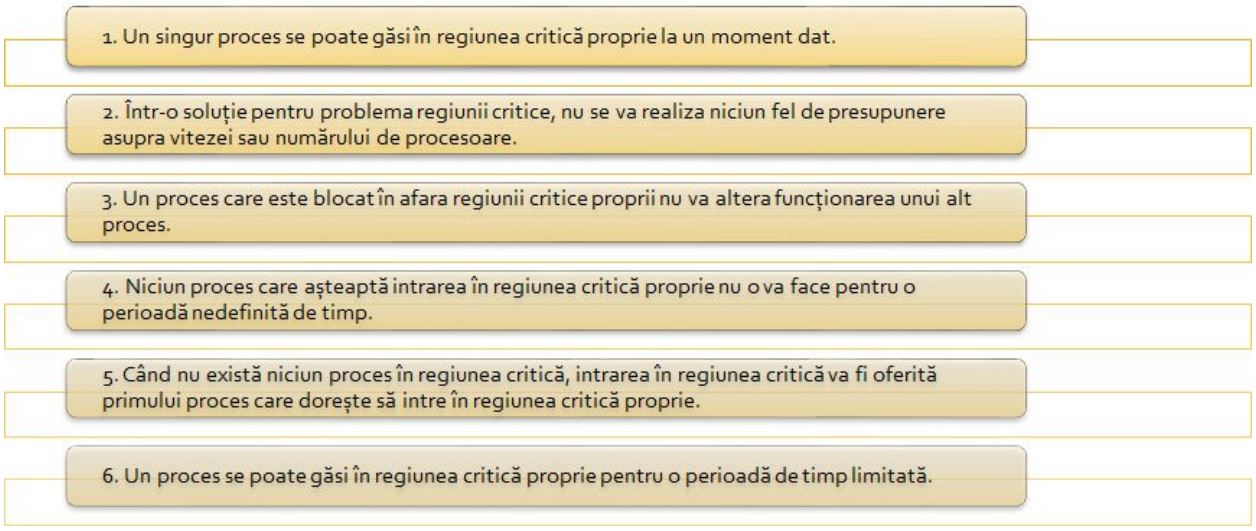
Condiţie de competiţie este o situaţie în care rezultatul execuţiei pentru două sau mai multe procese care partajează o serie de resurse depinde de ordinea de execuţie a acestora.

Regiune critică reprezinta partea unui proces în care sunt necesare servicii competitive (eventual din partea sistemului de operare, de regulă datorită unor resurse partajate).

Există o serie de condiții pentru o soluție corectă a problemei secțiunii critice:

1. Două procese nu se pot găsi simultan în interiorul regiunilor critice proprii.
2. Într-o soluție pentru regiunea critică, nu se va realiza nici un fel de presupunere legată de viteza sau numărul de procesoare.
3. Un proces care funcționează în afara regiunii critice proprii nu poate bloca activitatea niciunui alt proces.
4. Un proces nu va aștepta la nesfârșit intrarea în regiunea critică proprie.

Totuşi, există o a doua versiune a acestui set de condiţii (vezi W.Stallings) :



Protecţia regiunilor critice se poate realiza utilizând mecanisme specifice, capabile să asigură excluderea mutuală a proceselor.

Propuneri pentru obţinerea excluderii mutuale:

1. Dezactivarea întreruperilor
2. Variabile LOCK
3. Alternarea strictă
4. Soluţia lui Peterson
5. Instrucţiunea TSL

Blocarea întreruperilor

- Cea mai simplă soluţie care poate fi avută în vedere.Fiecare proces care intră în regiunea critică proprie trebuie să blocheze (deactiveze) întreruperilor.
- Nu va fi posibilă nicio întrerupere din partea altor procese. Această situaţie se referă la orice întrerupere, inclusiv întreruperea de ceas!
- Mai mult, procesorul nu mai poate fi alocat altor procese acum (prin dezactivarea întreruperii de ceas). Procesul curent este capabil să-şi finalizeze sarcinile fără interferenţa altor procese.

Variabile LOCK

Aceasta este o soluţie software simplă, bazată pe următoarea idee:

- Un proces care doreşte să intre în regiunea critică proprie va testa întâi valoarea unei variabile partajate, LOCK;
- Dacă valoarea este 0, o setează pe 1, şi intră în regiunea critică;

- Altfel, procesul va aştepta până când valoarea LOCK ajunge la 0.

Alternarea strictă: Aceasta este o versiune îmbunătăţită pentru soluţia precedentă, folosind variabile LOCK.

Soluţia lui Peterson: Această soluţie a fost, de fapt, oferită de către Dekker şi Peterson, ca o îmbunătăţire semnificativă a abordărilor precedente.

- De această dată procesele vor fi capabile să-şi paseze controlul regiunii critice prin utilizarea unei informaţii noi: variabila interest.
- Atunci când un proces deţine variabila turn, şi nu este interesat în utilizarea regiunii critice, un alt proces ar putea intra totuşi în regiunea critică (printr-o exprimare explicită a interesului).
- Totdeauna interesul pentru pătrunderea în regiunea critică este anunţat înaintea secţiunii critice, şi este ”uitat” odată cu părăsirea acesteia.

Instrucţiunea TSL (Test, Set, Lock): De această dată este oferită o soluţie hardware, bazată pe o instrucţiune oferită de o serie de procesoare. Sintaxa acestei instrucţiuni, TSL, este următoarea: TSL Reg, LOCK. Modul de acţiune este simplu:

- Este depozitată întâi valoarea din LOCK în registrul indicat (Reg);
- Apoi este depozitată o valoare ne-negativă în variabila LOCK.
- Aceste operaţii se realizează atomic (indivizibil), iar instrucţiunea TSL oferă această garanţie. Prin urmare, niciun alt proces sau procesor nu va putea accesa şi modifica locaţiile de memorie asociate instrucţiunii TSL.

Primitivele Sleep() şi wakeup()

- Primitiva sleep() este utilizată pentru a bloca activitatea unui proces care aşteaptă intrarea în regiunea critică, în locul unui ciclu de aşteptare activă.
- Primitiva wakeup() va fi utilizată pentru a ”trezi” procesele care au fost blocate anterior.

Semafoarele au fost introduse de către Dijkstra pentru a evita apariţia unor semnale wakeup pierdute (vezi situaţia descrisă anterior). Semafoarele sunt utilizate pentru a ”contoriza” apelurile wakeup care au fost realizate.
Un semafor este o valoare întreagă ne-negativă, împreună cu două operaţii de bază: down () şi up () (P şi V, în notaţia originală, din Olandeză).

- Operaţia down verifică valoarea semaforului. Dacă aceasta este diferită de 0 va fi decrementată şi procesul poate trece. Dacă valoarea semaforului este 0, procesul este blocat până când operaţia down iniţiată se poate încheia.

- Operaţia up este utilizată doar pentru a incrementa valoarea asociată unui semafor. Ambele operaţii sunt garantate ca fiind indivizibile, şi se pot desfăşura în siguranţă, fără ca un alt proces să intervină pe parcurs.

Mutex

- Ideea acestui mecanism se bazează pe semafoarele binare (deseori aceste semafoare sunt utilizate chiar sub numele de mutex).
- Variabilele Mutex sunt construite cu două stări: blocked şi free (unblocked). Implementarea de bază este similară implementării instrucţiunii TSL.
- Totuşi, spre deosebire de TSL, în implementarea variabilelor mutex este evitat ciclul de aşteptare activă.
- Sunt oferite două proceduri de bază, similare în construcţie cu procedurile oferite pentru TSL: mutex_lock şi mutex_unlock (vezi enter_region şi exit_region).
- Variabilele Mutex şi soluţiile bazate pe acestea sunt ideale pentru modelarea controlului secţiunilor critice, precum şi pentru aplicaţii multi-fir.

C.A.R Hoare şi B. Hansen au propus primitive de sincronizare puternice, de nivel înalt, cu implementare într-o serie de limbaje de programare concurente. **Aceste primitive poartă numele de monitoare.**

Un monitor este o colecţie de proceduri, variabile şi structuri de date, grupate împreună în module. Procesele pot apela procedurile din aceste module (monitoare) fără a exista acces direct către structurile interne.

Variabile de condiţie: Aceste situaţii pot fi protejate printr-un mecanism nou: variabilele de condiţie. Aceste structuri speciale sunt implementate împreună cu două operaţii simple: wait şi signal.

Transmiterea de mesaje

- Acest mecanism are la bază două proceduri simple: send() şi receive(). Există diferite metode de implementare, folosind metode distincte de identificare a părţilor comunicante.
- De exemplu, primitiva send() ar putea fi utilizată pentru a trimite mesaje către o destinaţie fixă, sau receive() ar putea fi utilizată pentru a aştepta mesaje din surse diferite.

Cerinţe

Un sistem bazat pe transmiterea de mesaje va satisface o serie de cerinţe care să asigure o implementare corectă:

- Stabilirea unui protocol pentru confirmarea recepţiei mesajelor. Acesta este necesar deoarece există un permanent pericol ca mesajele să poată fi pierdute. Fără niciun fel de confirmare, emiţătorul trebuie să retrimită mesajul.
- Evitarea mesajelor multiple. Atunci când se foloseşte protocolul anterior este posibilă apariţia mesajelor multiple.
- Numirea proceselor. Prin utilizarea acestui mecanism se poate evita ambiguitatea în identificarea părţilor comunicante. Această problemă/cerinţă este strâns legată de autentificarea proceselor.

CURS 12-13:

Orice sistem de calcul ofera diferite categorii de dispozitive de memorare:

- O cantitate relativ mica de memorie cache;
- O cantitate rezonabila de memorie (RAM), numita memoria principala;
- O cantitate semnificativa de memorie non-volatila, utilizata pentru depozitarea informatiei pe termen lung.

Managerul memoriei are ca principala sarcina gestiunea ierarhiei de dispozitive de memorare:

- Contabilizarea memoriei utilizate;
- Alocarea de memorie catre procese;
- Recuperarea memoriei eliberate;
- Gestionarea mecanismelor de baza de swapping, atunci cand este cazul

Categorii de sisteme de gestiune a memoriei:

- Sisteme care realizeaza o permanenta comutare a proceselor intre memoria principala si unitatile de disc:
    - Sisteme bazate pe paginare si swapping.
- Sisteme simple, fara comutare.

Monoprogramare - Presupune existenta unui singur program in memorie la un moment dat, impreuna cu sistemul de operare;
De-a lungul timpului au fost utilizate trei scheme de baza de organizare a memoriei pentru aceasta situatie:

- a) Schema utilizata pentru sisteme de operare mainframe si minicalculatoare.
- b) Schema folositapentru sistemede tip palmtop si embedded. Este posibil sa fie inca utilizata .
- c) Schema utilizata initial pentru micro-computers.

Multiprogramarea aduce cu sine doua probleme importante: protectia si relocarea.

- O posibilitate de a rezolva relocare presupune modificarea adreselor instructiunilor pe masura ce acestea sunt incarcate in memorie.
- O a doua varianta presupune divizarea memoriei in blocuri de cate 2K, fiecare bloc fiind inzestrat cu un cod de protectie pe 4 biti, pastratin PSW.
    - Protectia este asigurata prin acest mecanism datorita faptului ca doar sistemul de operare este capabil sa modifice codurile de protectie.
    - Orice incercare de acces la un bloc de memorie cu cod de protectie diferit este sanctionata printr-o instructiune trap. (OS360)

Swapping

- Pentru sistemelecu prelucrare in loturi, o organizare a memoriei simpla, bazata pe partitii de memorie fixa, este suficienta pentru a garata executia lucrarilor si pentru a mentine un grad inalt de ocupare al procesorului.
- Un sistem cu partajarea timpului nu este capabil, de regula, sa ofere suficiente resurse pentru a mentine permanent toate procesele in memorie.
- Pentru un astfel de sistem, o parte a lucrarilor vor fi depozitate temporar pe disc, urmand sa fie readuse in memorie intr-o maniera dinamica.

Sunt avute in vedere doua strategii diferite:

- 1. Procesele sunt depozitate sau aduse in memorie in intregime. Aceasta strategie se numeste **swapping**.
- 2. Procesele pot fi executate chiar daca nu se gasesc integral in memorie. Este vorba despre **memoria virtuala**.

1. In cazul **swapping-ului**, marimea, pozitia si numarul de partitii se modifica dinamic, pe masura ce procesele intra sau parasesc sistemul.
2. Dispar problemele legate de dimensiunile partiilor, insa apar probleme suplimentare legate de alocarea memoriei si urmarirea utilizarii acesteia.
3. **Swapping-ul** poate duce in timp la aparitia unui numar mare de goluri de dimensiuni relativ mici.

Tehnicile de compactare au fost dezvoltate pentru a permite crearea unui numar mic de goluri de dimensiuni relativ mari in locul unui numar mare de goluri de dimensiuni mici. Fiind mari consumatoare de timp, tehnicile de compactare sunt utilizate doar in situatii limita.

Bitmaps.

Hartile de biti ofera o metoda simpla de urmarire a modului in care este utilizata memoria.

- Memoria este divizata in unitati de alocare.
- Fiecarei unitati de alocare i se asociaza un bit, precizand daca unitatea de alocare este libera sau ocupata.

Harta de alocare este pastrata in memorie, prin urmare dimensiunea unitatii de alocare devine extrem de importanta.

- Ex. La 4 octeti, harta de alocare ocupa 1/33 din memorie.

O unitate de alocare mica va presupune o harta de dimensiuni relativ mari; O unitate de alocare mare duce la pierderi mari de memorie, avand in vedere faptul ca spatiul neutilizat intr-o unitate de alocare ar putea fi relativ mare.

O decizie de aducere in memorie a unui proces de dimensiune egala cu k unitati de alocare va presupune cautarea unei secvente de k biti nesetati in harta de alocare. Deoarece aceasta este pastrata in mai multi octeti consecutivi, operatia de cautare nu este triviala.

O a doua metoda de urmarire a alocarii memoriei presupune utilizarea unei liste inlantuite de segmente de memorie alocate si libere (numite si goluri).

- Pentru fiecare zona de memorie se pastreaza adresa de start si marimea, iar lista este sortata dupa adresa de start.
- Eliberarea unei zone de memorie va duce la crearea unei zone libere de cel putin aceeasi dimensiune, prin eventuala combinare cu zone libere alaturate.
- Pentru alocarea de memorie catre procese vor fi folositi diferiti algoritmi:

Fragmentarea memoriei:

Fragmentarea interna este o problema care apare in cazul partitiilor de dimensiune fixa. In aceasta situatie, programele de cele mai multe ori nu vor ocupa intregul spatiu al partitiei. Fiecare partitie va duce cu sine o cantitate de spatiu neutilizat.
Problema spatiului intern al unei partitii ramas neutilizat ca urmare a dimensiunii prea mari a partitiei fata de necesarul de memorie a programului se numeste **fragmentare interna**.

Situatia similara aparuta in cazul partitiilor dinamice se datoreaza golurilor numeroase de dimensiunirelativmici ramase in urma operatiilor de alocare.

• Problema golurilor aflate intre diferiteleprocese aflate in memorie poarta numele de fragmentareexterna. Memoria care nu este alocata devine in timp din ce in ce mai fragmentata.

• **Fragmentarea externa** poate fi eliminata prin tehnici costisitoare de compactare a memoriei.

Partitii gemene:

1. • Se presupuneca blocurilede memorie pot fi oferite in marimi de forma 2^k, intre limitele 2^L si 2^U.
2. • Intregul spatiu de memorie este un bloc de dimensiune2^U.
3. • O cerere pentru un spatiu intre2^U-1 si 2^U dFuce la alocarea intreguluibloc.
4. • Daca cererea este pentru un bloc de dimensiune mai mica, intregul spatiu este impartit in douablocuri identice de
5. dimensiune2^U-1.

6. • O cerere de dimensiuneintre 2^U-2 si2^U-1 este satisfacuta cu unuldintreaceste blocuri.
7. • Altfel, procesul de divizare continua pana la obtinerea unuibloc de dimensiune2^U-k, astfel incat 2^U-k-1<s<=2^U-k.
8. • La eliberarea unuibloc, acesta va fi compactat cu blocul lui geaman, daca ambele suntlibere.
9. • Algoritmul este utilizatuneoripentru sarcinile interne ale unor nuclee de sisteme de operareUNIX.

Chiar daca tehnicile bazatepe **swapping** sunt extrem de utile intr-un sistem cu multiprogramare, acestea nu pot oferi posibilitatea depasirii limitelorfizice ale memoriei.

Una dintre solutiile de inceput pentru depasirea limitarilorfizice impuse de memorie presupune impartirea programelorin mai multe bucati, **overlays**.

Executia unui program incepe, in aceasta situatie, cu prima bucata disponibila. Odata incheiata executia unei bucati, este incarcata de pe disc o alta bucata. • Sistemul de operare permitea pastrarea in memorie a unui numar mare de bucati-program (overlays), astfel incat sarcinile de gestiune si design ale acestora sunt extrem de dificil de realizat. • Sistemul de operare ofera suportul pentru aducerea si eliminarea din memorie, intr-o maniera dinamica, a segmentelor de program. • Impartirea programelorin segmente este realizata insade catre programator. Prin automatizarea sarcinilor de impartire a programelor in bucati (propusa initial de Fotheringham), au fost propuse primele tehnici de memorie virtuala. In aceasta situatie, sistemul de operare va determina singur daca o aplicatie urmeaza sa fie impartita in bucati si va determina modul in care se poate realiza aceasta impartire. Intr-un sistem cu multiprogramare, un proces care asteapta ca un segment sa fie adus in memorie este considerat ca fiind in asteptarea unei operatii de intrare/iesire.• Paginarea este o tehnica de baza pentru sistemele bazate pe memorie virtuala. • Intr-un sistem de calcul exista posibilitatea generarii unei cantitati limitate de adrese. Aceste adrese pot depasi memoria fizica disponibila: ▫ Ex: pentru un sistem pe 16 biti este posibila generarea unor adrese pana la 64K. Pentru aceste sisteme era posibil ca memoria fizica sa fie limitata la doar 32K… • Adresele generate de aplicatii se numesc adrese virtuale de memorie, si formeaza spatiul adreselor virtuale, uneori diferit de spatiul adreselor fizice. Pentru a rezolva aceste diferente, adresele virtuale de memorie sunt mapate pe adresele fizice de memorie prin intermediul MMU.

Daca spatiul adreselor virtuale depaseste spatiul adreselor fizice, chiar daca este posibila construirea unei aplicatii care sa foloseasca intregul spatiu de adrese virtuale, aceasta nu va putea fi incarcata integral in memorie. • Spatiul adreselor virtuale este divizat de regula in mai multe pagini de adrese. Corespondentul acestora in memoria fizica se numesc pagini cadru. • Cererile de acces la memorie sunt rezolvate astfel: ▫ Este identificata intai pagina de adrese virtuala in care se gaseste adresa. ▫ Pentru aceasta este determinata pagina cadru pe care este mapata. ▫ Adresa este calculata de MMU relativ la adresa de inceput a paginii cadru, folosind offset-ul din pagina de adrese.

Structura intrarilor. Urmatoarele informatii sunt depozitate in intrarile din tabelele de pagini:

- Bitul “caching”: inhiba caching-ul pentru pagina; util pentru sistemele care permit maparea unor pagini pe registrii unor dispozitive de intrare/iesire.
- Bitul de referinta: este setat de fiecare data cand are loc o referinta catre pagina. Este folosit pentru eliminarea paginilor din memorie.
- Bitul de modificare (dirty): folosit pentru a preciza daca pagina a fost modificata, in vederea salvarii continutului
- Biti de protectie: folositi pentru a preciza tipurile de acces permise.
- Bitul de prezenta Numarul paginii cadru.

1. Primele sisteme de calcul erau caracterizate prin:
   - cartelă pentru controlul lucrărilor
   - sisteme mono-utilizator

2. Un sistem de operare este:
   - un alocator de resurse
   - un program de control
   - o componentă a sistemului de calcul

3. Spooling-ul permite:
   - utilizarea simultană a periferiilor
   - utilizarea sistemelor de discuri

4. Un sistem cu prelucrare în loturi multiprogramat permite: (p 19)
   - execuția simultană a mai multor lucrări
   - ca sistemul de operare să ia decizii în numele utilizatorilor
   - execuția secvențială a lucrărilor

5. Propoziții corecte: (despre memorie) (p 68)
   - memoria își pierde conținutul în lipsa alimentării calculatorului
   - memoria depozitează programele și datele unui sistem de calcul
   - procesorul nu este capabil să acceseze direct memoria principală (?)

6. Gestiunea proceselor pp: (din partea S.O)
   - asigurarea unor mecanisme de sincronizare
   - asigurarea unor mecanisme de comunicare

7. Propoziții corecte referitoare la interpretorul de comenzi: (p 52)
   - oferă interfață între utilizator și sistemul de operare
   - oferă un mediu pentru executarea programelor

8. Propoziții corecte referitoare la mașinile virtuale: (p 39)
   - sunt copii exacte ale mașinii fizice
   - oferă o protecție completă a resurselor sistemului
   - pot poseda un mod monitor și un mod utilizator.

9. Scopurile unui sistem de operare cuprind următoarele:
- utilizarea eficientă a resurselor
- oferirea resurselor de calcul de bază
- ușurința operării

10. Propoziții corecte:
- sistemul de op. este responsabil cu realizarea ops de intrare/ieșire
- regiștii bază și limită pot fi încărcați cu ajutorul unor instruct. privilegiate

11. Monitoarele rezidente servesc la: (p 14)
- succesiunea automată a lucrărilor
- interpretarea cartelelor de control

12. Terminarea proceselor poate avea loc:
- ca urmare a unui apel sistem exit
- ca urmare a unei situații de excepție

13. Caracteristicile sistemelor cu partajarea timpului (p 31)
- lucrările

??

14. Propoziții corecte:
- la apariția unei întreruperi, componente hardware transferă controlul către sistemul de operare
-

15. Sistemul de operare trb să: (p 46-51)
- asigure formatarea fișierelor

16. Propoziții corecte:
- instrucțiunile privilegiate se pot executa în modul monitor (p77 ??)
- o instrucțiune privilegiată în mod utilizator determină o întrerupere către sistemul de operare.

17. Crearea procesela poate avea loc:
- la pornirea calculatorului
- ca urmare a unei cereri utilizator
- ca urmare a unui apel sistem specific.

18. Tranziții posibile: (p 93-98)
- un proces în curs de execuție devine proces pregătit de execuție
- un proces în curs de execuție devine proces suspendat (?) (p 98?)
- un proces suspendat devine proces pregătit pt. execuție

19. Instrucțiuni ce ar tb. să fie privilegiate:
- fixarea unui timer
- citește ceasul (sistem) (?)
- dezactivează întreruperile

Notiuni

- Multiprogramarea (p 19)
- Multithreading = mai multe fire de exec partajează resurse logice ale unui proces în cadrul căruia se execută.
- Time-sharing = partajarea timpului procesor între procese: fiecare proces va rula o cuantă de timp.
- Spooling (p 17)
- Proces = program în curs de execuție + date necesare (p 89 - sus)
- Fir de execuție = extensie a modelului clasic de proces
- Fișier = unitatea logică de depozitare a informației într-un sistem de calcul
- Interpretor de comenzi = program sistem ce oferă o interfață de bază a utilizatorului cu SO; face parte din nucleul SO; oferă suportul pt execuția altor programe.
- Semafor = variabila întreagă în care se contorizează "treziile" din utilizările anterioare. (p 134)

Definitii:

- Adresele ce pot fi generate de un program formează **spațiul de adrese virtuale**. **MMU (Memory Management Unit)** mapează adresele virtuale în spațiul adreselor fizice.
- **Monoprogramarea** este o tehnică de exploatare pentru sistemele seriale. Obiectivul acestei tehnici este automatizarea lansării în execuție a lucrărilor (programelor). Pentru aceasta, lucrările sunt organizate secvențial, în loturi de lucrări şi lansate automat în execuție.
- **Multiprogramarea** este tehnica de exploatare a sistemelor de calcul care permite existența simultană în memoria internă a mai multor programe care se execută concurent, în partiții fixe de memorie, cu restricția ca ele să nu folosească în acelaşi timp aceeaşi resursă.
- Tehnica **SPOOLING** (Simultaneus Peripheral Operations On-Line) reprezintă un mod eficient de exploatare a sistemelor de calcul seriale, bazat pe principiul separării operațiilor de intrare de operațiile de ieşire şi de restul prelucrărilor şi pe executarea lor în paralel.
- Alocarea paginată se realizează astfel:
    - memoria internă se împarte în zone de lungime fixă, numite pagini fizice
    - memoria virtuală alocată unui proces se împarte în zone de aceeaşi lungime cu paginile fizice, numite pagini virtuale
    - pentru fiecare proces în execuție, care utilizează memoria virtuală, se crează o tabelă de pagini; în această tabelă se memorează numărul paginii fizice în care a fost încărcată fiecare pagină virtuală;
- Se spune că un set de procese este în **deadlock** dacă fiecare proces din set aşteaptă un eveniment care poate fi generat doar de un alt proces din set. În cele mai multe din cazuri, evenimentul aşteptat de fiecare proces reprezintă eliberarea unei resurse de către un alt proces din sistem.
- **resurse preemptibile** – o astfel de resursă poate fi luată de procese prioritar fără efecte negative asupra procesului;
- **resurse nepreemptibile** –o resursă nepreemptibilă nu poate fi luată de la proces prioritar fără a perturba procesul.

Algoritmul ceasului pentru paginare:

Un algoritm asemanator Second-chance este **Clock Relacement(First In, Not First Used Out)**. Ca si primul, el provine din FIFO; practic, se obtine acelasi rezultat ca si la Secondchance, doar implementarea fiind cea care difera. Ceea ce schimba algoritmul Clock este incovenientul pe care il prezinta Scond-chance: mutarea paginilor in interiorul listei. El evita acest lucru folosind o lista circulara ce contine paginile si un pointer care indica cea mai veche pagina din memorie. Cand se ajunge la o eroare de pagina, algoritmul verifica bitul R al paginii indicate. Daca acesta are valoarea 0, pagina este eliminata si in locul ei este incarcata o noua pagina in memorie, pointerul mutandu-se la pagina uramtoare din lista; daca bitul R=1, acesta este resetat, iar pointerul avanseaza pana cand gaseste o pagina cu R=0 care poate fi inlocuita.