Examene SO

1. Structura sistemelor de operare
2. 2 exemple de organizare a structurii sistemelor de operare
3. Monolotic: Intr un sistem de operare monolitic, intregul nucleu (kernel) al sistemului de operare functioneaza ca o singura unitate monolitica. Toate functionalitatile, cum ar fi gestionarea proceselor, sistemul de fisiere si gestionarea memoriei, sunt implementate intr un singur bloc de cod.

Avantaje: Acces rapid la toate functionalitatile, performanta buna pentru operatiunile de baza, complexitate redusa in dezvoltare

Dezavantaje: Dificultati in mentinerea si extinderea codului, o problema in cazul erorilor care pot afecta intregul sistem

1. Microkernel : intr un system de operare bazat pe microkernel, functionalitatile de baza cum ar fi gestionarea proceselor si comunicarea intre procese, sunt implementate intr un nucleu mic (microkernel), iar restul componentelor sunt implementate ca procese separate care ruleaza in spatiul utilizatorului

Avantaje: Flexibilitate mare, usurinta in dezvoltarea si extinderea sistemului, izolare mai buna intre componente

Dezavantaje: Performanta potential mai scazuta decat in sistemele monolitice, din cauza costului suplimentar de comunicare intre procese

1. Pe scurt diferentele dintre cele doua:
   1. Arhitectura
      1. Monolitic: toate functionalitatile sunt implementate intr un singur nucleu monolitic
      2. Microkernel: functionalitatile de baza sunt implementate intr un nucleu mic, iar restul componentelor ruleaza in spatiul utilizatorului
   2. Performanta
      1. Monolitic: Poate oferi o performanta mai buna pentru operatii de baza, datorita accesului direct la toate functionalitatile
      2. Microkernel: Poate suferi de performanta mai scazuta din cauza costului suplimentar de comunicare intre procese
   3. Extensibilitate si Mentenabilitate
      1. Monolitic: Dificultati in mentenanta si extensibilitate, deoarece modificarile pot afecta intregul system
      2. Microkernel: Mai usor de extins si de mentinut, deoarece componente suplimentare pot fi adaugate sau schimbate fara a afecta nucelul mic
   4. Izolare
      1. Monolitic: Mai putina izolare intre functionalitati, astfel incat o problema undeva poate afecta intregul sistem
      2. Microkernel: Izolare mai buna, deoarece functionalitatile sunt separate in procese independente
   5. Complexitate
      1. Monolitic: Mai putina complexitate in dezvoltare, dar dificultati in gestionarea codului pe masura ce sistemul creste in dimensiune
      2. Microkernel: Complexitate mai mare la inceput, dar usurinta in gestionarea si extinderea codului pe termen lung
2. Exceptii si intreruperi
   1. Ce sunt exceptiile? 2 exemple

Exceptiile sunt situatii care apar in timpul executiei unui program sau a unui proces si care intrerup fluxul normal de executie. Acestea pot fi generate de SO, de hardware sau de software. Exceptiile pot fi de diferite tipuri, in functie de problemele care le cauzeaza:

-exceptii de divizare la zero: apare atunci cand un program incearca sa imparta un numar la 0, CPU detecteaza conditia si declanseaza aceasta exceptie

-exceptii de acces la memorie: apare atunci cand un program incearca sa acceseze o zona de memorie la care nu are acces. De ex daca un program incearca sa citeasca/scrie intr o zona de memorie rezervata SO

b. Rolul intreruperilor in sistemul de calcul

Ele sunt mecanisme care permit SO sa raspunda la evenimente asincrone, adica evenimente care pot aparea in orice moment, independent de fluxul curent de executie. In principal, au rolul de a permite executia prioritara a unui proces critic, cum ar fi sesizarea unui defect si prevenirea acestuia. De ex, dc un dispozitiv hardware detecteaza o eroare, poate genera o intrerupere care suspenda temporar procesul curent si permite sistemului de operare sa trateze problema.

c. Ce face kernelul la o intrerupere hardware. Descrieti mecanismul de tratare a intreruperii

Intreruperile hardware sunt semnale asincrone sau sincrone de la un periferic care semnalizeaza aparitia unui eveniment care trebuie tratat de procesor.

Pasi tratare intrerupere:

1. Detectarea intreruperii: procesorul detecteaza semnalul si intrerupe executia curenta
2. Salvarea starii curente: Salvarea registrului contor de program, a registrilor de stare, si a tuturor variabilelor din program care sunt afectate de executia rutinei de tratare a intreruperii
3. Executia rutinei de tratare a intreruperii: secventa de cod care trateaza evenimentul specific care a cauzat intreruperea
4. Restaurarea starii sitemului: restaurarea registrilor si variabilelor salvate la pasul 2
5. Continuarea executiei
6. Sincronizarea proceselor
   1. Ce este un monitor? Care sunt avantajele folosirii monitoarelor

Monitor = metoda de sincronizare a doua sau mai multe sarcini(procese) care folosesc o resursa comuna, de obicei un dispozitiv hardware sau o multime de variabile. Un monitor e compus din:

1. Set de proceduri ce permite interactiunea cu o shared resource

2. Mecanism sau “lock” pt mutual exclusion care asigura ca doar un proces poate accesa resursa la un moment dat

3. Variabile asociate resursei comune

4. Un invariant al monitorului ce defineste conditiile necesare evitarii unui conflict de interes

**Avantaje**:

* + - * siguranta: lock-ul e adaugat de compilator deci e mai sigur decat sa fie pus de programator
      * evitare busy waiting
      * Eficienta si productivitate
      * Accesul la informatii
      * Economii de costuri
      * Conectivitate globala
  1. Ce sunt variabilele conditie si cum se folosesc?

Sunt un mecanism de sincronizare utilizat in programarea concurenta. Ele sunt folosite pentru a permite unui proces sa astepte pana cand o anumita conditie este indeplinita. Sunt adesea asociate cu un mutex pentru a evita race conditions. Un proces care asteapta o conditie va elibera mutexul si va intra intr o stare de asteptare. Cand un alt proces modifica starea astfel incat conditia devine adevarata, acesta semnalizeaza variabila de conditie si apoi procesul se trezeste din asteptare si isi reia executia.

1. Deadlock
   1. Care sunt conditiile necesare producerii deadlock-ului?

Deadlock este o situatie in care 2 sau mai multe procese asteapta reciproc pentru resursele detinute de celalalt, rezultand intr o asteptare infinita.

4 conditii necesare pentru producere deadlock:

1. Mutual exclusion: o resursa poate fi detinuta de un singur proces la un moment dat
2. Hold and wait: un proces poate detine o serie de resurse in acelasi timp si poate solicita alte resurse care sunt detinute de alte procese
3. No preemption: o resursa nu poate fi preluata fortat de la un proces de catre un alt proces
4. Circular wait: exista un set de procese de asteptare in care fiecare proces asteapta o resursa care este detinuta de urmatorul proces din set si tot asa, formand un ciclu.
   1. Daca toate cele 4 conditii sunt indeplinite simultan => deadlock
5. Planificarea proceselor
   1. Ce este invesiunea de prioritati?

Un fenomen care apare in SO cu planificare bazata pe prioritati. Acesta se produce atunci cand un proces cu o prioritate mai mica detine o resursa necesara unui proces cu prioritate mai mare => “inversarea” prioritatilor

* 1. De ex: Sa presupunem ca avem 3 procese: P1, P2, P3, unde P1 are cea mai mare prioritate (sa spunem 2, considerand un numar cat mai mic o prioritate mai mare), P2 sa spunem ca are prioritate 5, iar P3 are prioritate 8. Daca P3 detine o resursa necesara pentru P1 si P2 este gata sa ruleze, SO va alege sa ruleze P2 deoarece P1 nu poate rula fara resursa detinuta de P3. Astfel, P3 care are cea mai mica prioritate impiedica P1 sa ruleze, desi are cea mai mare prioritate
  2. O solutie:

Mostenirea prioritatii: Daca un proces cu prioritate inalta (P1 in exemplul dat) este blocat de un proces cu prioritate mai mica (P3) care detine resursa necesara, atunci P3 “mosteneste” prioritatea lui P1 pana cand se elibereaza resursa. Astfel, P3 va rula cu prioritatea lui P1, prevenind astfel ca alte procese cu prioritate intermediara sa il blocheze. Dupa ce P3 elibereaza resursa si P1 poate continua, P3 isi reia prioritatea initiala.

1. Alocare proportionala a resurselor
   1. Descrieti algoritmul planificarii tip loterie (Lottery Scheduling LS)

E un algoritm probabilistic de planificare a proceselor intr-un SO. Fiecare proces este atribuit cu un anumit numar de bilete de loterie, iar planificatorul alege un bilet aleator pentru a selecta urmatorul proces. Distributia biletelor nu trebuie sa fie uniforma; atribuirea unui numar mai mare de bilete unui proces ii ofera o sansa relativ mai mare de a fi selectat.

* + - * 1. Fiecare proces primeste un anumit nr de bilete de loto
        2. Planificatorul alege un bilet aleator
        3. Procesul care detine biletul e ales pentru executie
  1. Exemplu pentru 5 procese care detin 4, 6, 2, 5, 3 tichete

Fie P1, P2, P3, P4, P5 – procesele

Presupunem:

P1: bilete 1-4

P2: bilete 5-10

P3: bilete 11-12

P4: bilete 13-17

P5: bilete 18-20

Planificatorul alege aleator un nr intre 1 si 20.

Sa presupune ca numerele extrase sunt 7, 3,15, 11, 19. Atunci ordinea de executie va fi: P2, P1, P4, P3, P5.

* 1. Complexitatea algoritmului:

Complexitatea timp pentru selectarea urmatorului proces de executie este proportionala cu numarul de bilete = nr total de procese in cazul in care fiecare proces are un singur bilet)

* 1. Imbunatatire la O(logn):

Daca utilizam un arbore de cautare echilibrat, reducem timpul la O(logn), unde n = nr de bilete, iar fiecare nod va contine procesul si multimea de bilete pe care o detine

1. Gestiunea memoriei
   1. Descrieti mecanismul generic de suport hardware pentru protectia memoriei

Include mai multe componente si tehnici:

1 Registri pt baza si limita (orice incercare de accesare a memoriei in afara acestei limite => eroare)

2 Paginare si segmentare: Fiecare pagina sau frame are propriile atribute de protectie care pto fi verificate de hardware la fiecare accesare a memoriei

3 Unitatea de gestionare a memoriei (MMU) (hardware care traduce adresele de memorie virtuala in adrese fizice)

4 Liste de acces la memorie: Structuri de date care contin informatii despre drepturile de acces la memorie pentru fiecare proces

* 1. Ce este swappingul si la ce ajuta

Mecanism de gestionare a memoriei utilizat in SO. Implica mutarea datelor intre RAM si spatiul de stocare secundar (disk) pentru a elibera memorie pentru alte procese

Ajuta la: maximizarea utilizarii CPU, gestionarea memoriei, prevenirea supraincarcarii memoriei

* 1. 3 Exemple de alocare dinamica pentru gestiunea holes + cum functioneaza:

1 First Fit: aloca memoria primului hole care e suficient de mare pentru a incapea procesul

2 Best Fit: cauta prin toate “holes” disponibile si aloca memoria celui mai mic hole suficient de mare

3 Worst Fit: aloca memoria celui mai mare “hole: indiferent de dimensiunea procesului

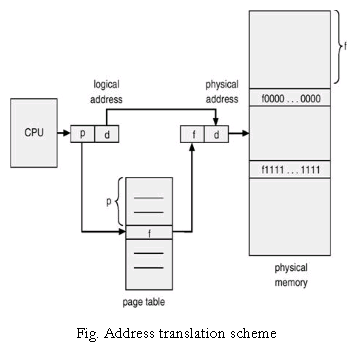
* 1. Ce e fragmentarea memoriei si de cate tipuri e?

Fenomen care apare atunci cand memoria e alocata si dealocata in mod repetat => divizarea memoriei in blocuri mici

1 Fragmentarea interna: blocul de memorie alocat unui proces e mai mare decat cantitatea solicitata de proces (i.e ultimii bytes din pagina nu se folosesc)

2 Fragementarea externa: memoria libera este impartita in blocuri mici si dispersate. In timp, pe masura ce procesele sunt incarcate si descarcate in si din memorie, spatiul liber poate deveni “fragmentat”. Adica, memoria libera totala poate fi suficienta pentru a satisface o cerere dar nu este contigua; e fragmentata intr o serie de blocuri de spatiu liber separate.

* 1. Transformarea adreselor logice in adrese fizice



* 1. Rezolva paginarea vreo problema a fragmentarii memoriei?

Da, rezolva fragmentarea externa a memoriei. Memoria e impartita in blocuri de dimensiune fixa numite pagini. Atunci cand un proces are nevoie de memorie, SO aloca un numar intreg de pagini pentru el => elimina fragmentarea externa (nu exista spatiu liber intre pagini (frame-uri pentru ca vorbim de memoria fizica), dar creeaza fragmentare interna; pentru o pagina de exemplu de 128 bytes, daca un proces are nevoie de 98 de bytes, SO aloca o intreaga pagina => 30 de bytes alocati inutil => fragmentare interna

1. Memoria virtuala
   1. Cum functioneaza demand paging?

E o tehnica de gestionare a memoriei pentru optimizarea resurselor. Se copiaza o pagina de pe disk in memoria fizica doar daca se face o incercare de a o accesa si pagina nu e deja in memorie (se produce page fault)

1) Se cauta pagina in page table

2) nu se gaseste

3) se genereaza page fault => page table trimite semnal catre SO

4) Paginare la cerere

- se incarca pagina necesara din stocarea secundara in memorie

- daca sistemul nu are suficienta memorie libera trebuie sa selecteze o pagina “victima” pentru a fi evacuata din memorie pentru a face loc paginii de care avem nevoie

- SO actualizeaza page table pentru ca s-au realizat modificari la nivelul sau

- se aloca memorie pentru fiecare proces (se reia executia care a dus la page fault acum avand pagina in memorie)

* 1. Avantaje

1. Maximizarea utilizarii memoriei: se incarca doar paginile necesare pentru executie => eficientizarea memoriei

2. Reducerea timpului de initializarea al proceselor: deoarece sunt incarcate doar paginile necesare doar atunci cand e nevoie

3. Eliminarea fragmentarii externe: paginile au aceeasi dimensiune => nu exista spatiu liber intre pagini, dar exista fragmentare interna

4. Partajarea memoriei: 2 sau mai multe pagini pot vedea aceeasi zona de memorie incarcand paginile logice in acelasi cadru fizic

* 1. Ce este page-fault si cum e tratat?

E o intrerupere care se produce atunci cand un program incearca sa acceseze o pagina de memorie care nu este curent in memorie.

Tratarea de catre kernel:

1. SO verifica daca adresa de memorie la care programul incearca sa acceseze este valida. Daca e invalida atunci termina programul

2. Daca adresa e valida, dar pagina nu este in memoria fizica, SO incepe procesul de incarcare a paginii in memoria fizica. Acest lucru poate implica evacuarea unei alte pagini pentru a-i face loc (in cazul in care memoria este plina)

3. Dupa ce e incarcata in memoria fizica, SO actualizeaza page table pentru a reflecta noua locatie a paginii

4. SO reia executia programului de la punctul in care a fost intrerupt

d. Effective Access Time – EAT Formula

EAT = , unde este timpul pentru TLB search, iar 1 reprezinta 1\*timpul pentru accesarea memoriei, si 2 = 2\* timpul pentru accesarea memoriei

e. Ce este tehnica Copy-On-Write (COW) si la ce foloseste?

Tehnica Copy-On-Write, uneori denumita si partajare implicita, este o tehnica de gestionare a resurselor utilizata in programarea computerelor pentru a implementa eficient o operatie de “duplicare” sau “copiere” pe resurse modificabile. Aceasta este folosita cel mai adesea in gestionarea memoriei virtuale. Ideea fundamentala este ca daca mai multi apelanti solicita resurse care initial sunt de nedistinguit, li se pot oferi pointeri catre aceeasi resursa. Aceasta functie poate fi mentinuta pana cand un apelant incearca sa modifice “Copia” sa a resursei, moment in care se creeaza o copie privata reala pentru a preveni ca modificarile sa devina vizibile pentru toti ceilalti. Toate acestea se intampla in mod transparent pentru apelanti. Avantajul: daca un apelant nu face niciodata modificari, nu trebuie creata niciodata o copie privata.

f. Cate tipuri de algoritmi de inlocuire in functie de setul din care se alege victima exista? Explicati consecintele alegerii fiecaruia

Exista doua tipuri principale de algoritmi de inlocuire a paginii in functie de setul de frame-uri din care se alege victima:

1. Algoritm de inlocuire statici: aleg victima dintr un set fix de frame uri. De ex FIFO (First In First Out) inlocuieste pagina care a stat cel mai mult in memorie. Acesta poate duce la eliminarea unei pagini care este totusi folosita intens

2. Algoritmi de inlocuire dinamici/variabili: Pot alege victima dintr un set variabil de frame-uri: De ex NRU (Not Recently Used) care imparte paginile in 4 clase in functie de cand au fost referite sau modificate ultima data si alege o pagina aleator din clasa cu cea mai mica eticheta. Clasele:

(0,0) – nu a fost referita recent si nici modificata (Cea mai buna alegere pentru evacuare)

(0,1) – nu a fost referita recent dar a fost modificata

(1,0) – a fost referita recent dar nu a fost modificata (foarte probabil sa fie modificata in viitor)

(1,1) – a fost si referita recent si modificata (deci o pagina din aceasta clasa ar fi cea mai proasta alegere pentru evacuare)

NRU simplu si eficient de implementat si asigura performante acceptabile deoarece incearca sa nu elimine o pagina care a fost referita recent/modificata.

9) Alocarea memoriei Buddy System 4kb

A = allocate(53);

B= allocate(1536);

C = allocate(512);

D = allocate(1536);

E = allocate(512);

free(C);

free(A);

free(B);

free(E);

free(D);

Memorie intiala 4KB: 4\*1024b = 4096b

[0-4095] -liber

Alocam A:

[0-63] -A

[64-4095] – liber

Alocam B:

[0-63] – A

[64-2111] -B

[2112 – 4095] – liber

Alocam C

[0-63] – A

[64-2111] -B

[2112-2623] -C

[2624-4095]-liber

Alocam D

SO refuza pentru ca nu are spatiu

Alocam E

[0-63] – A

[64-2111] -B

[2112-2623]-C

[2624-3135] -E

[3135-4095] – liber

Free(C):

[0-63] – A

[64-2111] -B

[2112-2623]-liber

[2624-3135] -E

[3135-4095] – liber

Free(A):

[0-63] – liber

[64-2111] -B

[2112-2623]-liber

[2624-3135] -E

[3135-4095] – liber

Free(B):

[0-63] – liber

[64-2111] -liber

[2112-2623]-liber

[2624-3135] -E

[3135-4095] – liber

=>

[0-2623] – liber

[2624-3135] -E

[3135-4095] – liber

Free(E):

[0-2623] – liber

[2624-3135] -liber

[3135-4095] – liber

=>

[0-4095] – liber

10) Aplicatie concurenta filtrare de imagini ruleaza pe un multiprocesor simetric cu 4 procesoare pentru a filtra o imagine pe care o imparte in 4 cadrane, fiecare cadran – un procesor

a. Ce solutie pentru a implementa aplicatie ati alege 4 procese sau 4 threads? Explicati

-bazata pe 4 threads. Motive:

1. Partajarea memoriei: Threads din acelasi proces partajeaza spatiul de memorie al procesului => cele 4 threads pot accesa imaginea fara a fi nevoie de mecanisme costisitoare IPC. In cazul proceselor, fiecare proces are spatiul sau de memorie, ceea ce ar face partajarea imaginii intre procese mai dificila si costisitoare dpdv al performantei

2. Comunicare: intre threads e mai rapida si mai simpla de realizat decat cea dintre procese. Util daca exista necesitate de sincronizare/comunicare intre diferite parti ale imaginii

3. Eficienta: Crearea unui nou thread este, in general, mai putin costisitoare dpdv al resurselor decat crearea unui nou proces. Procesele au nevoie de un nou spatiu de adresa, pe cand threads pot rula in spatiul existent al procesului.

Procesele ar fi o alegere mai buna daca ar fi nevoie de fiabilitate (un crash intr un thread ar duce la crashul intregului proces, pe cand procesele ar fi izolate)

b. Amdahl’s Law

S = , S – nr procesoare, p – fractiunea de timp in care poate fi paralelizat

(1-p) = Tcp sau Tct

K = Tcp/Tct

S procese = = 1/Tcp=1/K\*Tct

S threads = = 1/Tct

S procese<S threads => versiunea cu threads va avea o imbunatatire mai mare

c. Presupunem ca nr de procese creste la N finit (multiplu de 4) Legea lui Gustafou in contextu astA?

- Legea lui Gustafon – alternativa la Amdahl ce ofera perspectiva mai optimista asupra scalabilitatii calculului paralel. Amdahl – problema fixa ce examineaza imbunatatirea performantei prin cresterea nr de procese

Gustafon – dimensiunea problemei creste odata cu nr de procese

-daca crestem dimensiunea problemei proportional cu nr de procesoare => imbunatatire liniara a performantei (Scalabilitate liniara)

Sa spunem ca N = 8 => putem imparti imaginea in 8 cadrane, unul pentru fiecare procesor, si asa facem pentru orice numar ar fi N finit, imparitm imaginea in N cadrane, cate unul pt fiecare procesor

d. Folositi legea lui Gustafon

Folosind legea explicata anterior => daca folosim threads in loc de procese => cost de creare mai mic => Tcp = K\*Tct > Tct

Examen 7 feb 2023

1. Termen Definitie

(i) Race condition (d) Acces intretesut care produce rezultate nedeterministe

(ii) Deadlock (a) Resursele necesare unui proces sunt detinute de altul si inv

(iii) Sectiune critica (e)Cod care acceseaza date partajate

(iv) Excludere Mutuala (b) Cel mult un proces acceseaza datele partajate

(v) Sincronizarea (f) evenimente care apar intr o ordine bine specificata

2. Definitii

(a) adresa virtuala: Adresa folosita intr un sistem de calcul care e folosita in cadrul unui sistem de gestionare a memoriei virtuale. Adresa generata de CPU atunci cand un proces incearca sa acceseze o locatie din memorie.

(b) adresa fizica: Adresa reala in memoria unui sistem de calcul. Reprezinta locatia efectiva in memoria hardware unde sunt stocate datele

(c) Spatiu de adrese virtuale: un interval de adrese virtuale pe care un proces le poate utiliza. E creat de SO atunci cand un proces este lansat si este unic pentru fiecare proces. Permite unui proces sa functioneze ca si cum ar avea acces la o regiune continua de memorie, chiar daca in realitate datele pot fi stocate in locatii neconsecutive in memoria fizica sau chiar pe disk

(d) Present (valid) bit: indicator folosit in gestionarea memoriei virtuale a unui SO. Face parte din intrarile din page table care mapeaza adresele virtuale la cele fizice. Acest bit indica daca o pagina specificata este prezenta in memoria fizica (RAM) sau nu. Daca e setat la 1 => e in Ram, daca e setat la 0 => nu e, poate fi stocata pe disk intr o zona numita swap space.

(e) Pagina vs frame: Ele apar in contextul gestionarii memoriei virtuale.

Pagina: unitate de date manipulata in memoria virtuala. Un program vede memoria ca o secventa continua de pagini. Atunci cand un program acceseaza o adresa de memorie, aceasta e divizata in numar de pagina si offset.

Frame: o pagina de memorie fizica; SO controleaza maparea dintre pagini si frame uri. Atunci cand un program acceseaza o pagina, SO traduce adresa paginii intr o adresa de frame pentru a localiza datele in memoria fizica.

->Segentarea memoriei: 3 diferente dintre paginare si segmentare

Segmentarea memoriei: tehnica de gestionare a memoriei in care memoria e impartita in segmente de dimensiuni variabile. Fiecare segment are un rol specific, sunt indexate printr-un tabel de segmente care stocheaza adresa de baza si limita fiecarui segment.

Diferente:

1) Dimensiunea: paginile au toate aceeasi dimensiune, la segmentare, memoria e impartita in segmente de dimensiuni variabile

2) organizarea datelor: in paginare datele sunt organizate intr un mod transparent pentru program. In segmentare, ele sunt organizate in segmente cu diferite semnificatii: segmente de cod, segmente de date etc

3) Gestionarea spatiului liber: in paginare spatiul e gestionat mai usor (toate paginile au aceeasi dimensiune); in segmentare: gestionarea spatiului liber poate fi mai complexa din cauza problemelor precum fragmentarea externa.

3. Gestiunea memoriei

a) Operatiile unui TLB, implicatii asupra context-switch-ului

TLB este o mica memorie cache hardware care stocheaza cele mai recente mapari de adrese virtuale la adrese fizice. Atunci cand un proces incearca sa acceseze o adresa de memorie, sistemul verifica mai intai in TLB pentru a vedea daca maparea exista deja acolo. Daca da, adresa fizica e returnata imediat => acces rapid la memorie. Daca nu, sistemul trebuie sa caute in page table => un proces mai lent

Impact asupra context-switch: atunci cand se produce un context-switch, maparile din TLB pot deveni invalide, pentru ca fiecare proces are spatiul sau de adrese virtuale => la fiecare context-switch trebuie actualizat si TLB => costisitor

b) Tabela inversata de pagini si cand se folosestE? Ce se intampla cand exista pagini partajate + solutii

Tabela inversata de pagini: structura de date folosita in gestionarea memoriei virtuale pentru a reduce cantitatea de memorie necesara pentru a stoca tabela de pagini. In loc sa aiba o intrare pentru fiecare pagina virtuala (ca intr un page table normal), are o intrare pentru fiecare frame din memorie.

Cand exista pagini partajate in sistem: deoarece fiecare intrare in tabela se refera la un singur frame din memorie, nu e clar cum sa gestionee situatia in care mai multe pagini virtuale din procese diferite sunt mapate la acelasi frame de memorie (pag partajata)

Solutii:

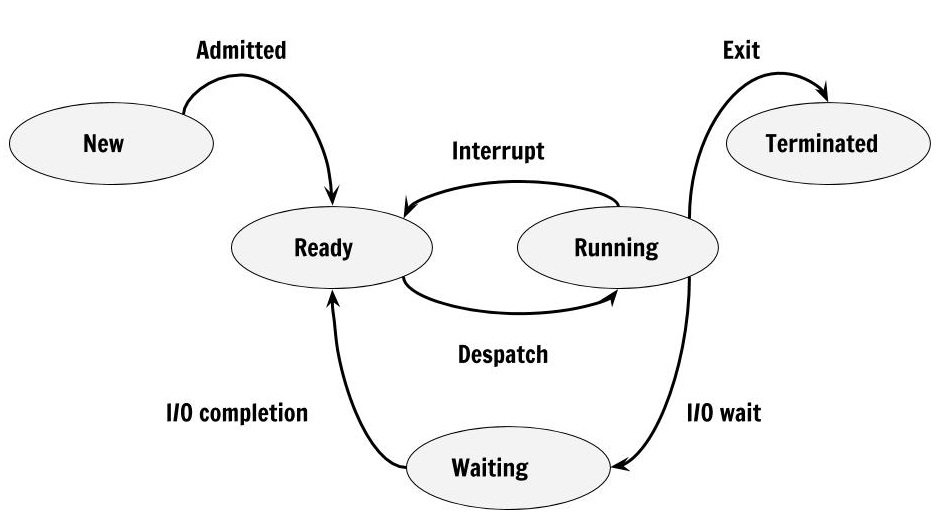
1. Intrari multiple in tabela inversata de pagini pentru acelasi frame de memorie, fiecare cu propriul context de proces si adrese virtuale => utilizarea ineficienta a memoriei

2. Tabele de pagini suplimentare

3. Liste de partajare: care contin informatii despre toate paginile virtuale care sunt mapate la un anumit frame de memorie

4. Controlul proceselor/threadurilor

-principalele stari ale unui proces in executie si explicati in ce conditii se face tranzitia intre ele



- apel vfork, precautii, necesitate:

Vfork() similar cu fork(), cu cateva diferente. Fork() creeaza un nou proces prin duplicarea spatiului de adresa al procesului parinte, vfork() creeaza un nou proces fara a copia page tables ale procesului parinte.

Precautii: comportamentul este nedefinit daca procesul creat de vfork() modifica orice date in afara unei variabile de tip pid\_t folosite pentru a stoca valoarea returnata de vfork(), sau daca se intoarce din functia in care a fost apelat vfork(), sau daca apeleaza orice alta functie inainte de a apela cu succes \_exit() sau una din functiile familiei exec()

Orice modificare a memoriei de catre procesul copil va fi vizibila si de catre procesul parinte, iar parintele e suspendat pana cand copilul se termina.

-2 avantaje thread kernel vs procese

1. Crearea si comutarea mai eficiente: crearea unui thread in cadrul unui proces existent este mai rapida si eficienta dpdv al resurselor decat crearea unui proces complet nou. Similar si pentru comutarea intre thread-uri fata de procese datorita costului ridicat pentru context-switch.

2. Partajarea resurselor: Threadurile din acelasi proces partajeaza spatiul de adrese al procesului si au acces direct la datele si resursele procesului => Faciliteaza comunicarea intre threads deoarece pot citi/scrie in aceeasi locatie de memorie. Pe de alta parte, procesele au spatii de adresa separate si necesita un mecanism IPC pentru a partaja datele.

-2 avantaje threads user vs cele kernel

1. Performanta mai buna in ceea ce priveste crearea si comutarea lor, pentru ca nu necesita interventia kernelului.

2. Control mai bun asupra planificarii pentru ca sunt gestionare de biblioteca de threaduri a pricesului, userul poate implementa propriul algoritm de planificare, pe cand cele kernel sunt planificate de SO si poate sa nu fie optimizat pentru modelul de executie al aplicatiei.

5. Stocarea datelor

-Ce e un i-node Unix si la ce foloseste?

E o structura de date care stocheaza informatii despre un fisier sau un director. Fiecare fisier sau director dintr-un sistem de fisiere Unix are un i-node asociat. Acesta contine metadate despre fisier, cum ar fi permisiunile de acces, proprietarul fisierului, grupul, dimensiunea, timpul de creare, ultima modificare si ultimul acces la fisier, dar si date despre locatia datelor fisierului pe disk.

Atunci cand un proces acceseaza un fisier, SO cauta i-node ul corespunzator pentru a determina unde sunt stocate datele fisierului si ce operatii sunt permise asupra lui

-Ce este un v-node si la ce foloseste

Virtual node: structura de date utilizata in unele siteme de fisiere pentru a reprezenta un fisier sau un director. Parte a unui strat de abstractizare intre SO si sitemul de fisiere fizic.

Fiecare v-node contine un set de operatii care pot fi efectuate pe fisierul/diretorul pe care-l reprezinta

Utilizarea v-node urilor permite SO ului sa interactioneze cu sisteme de fisiere diferite intr un mod uniform. Atunci cand SO vrea sa efectueze o operatie pe un fisier, el apeleaza operatia corespunzatoare din v-node fara a avea nevoie sa stie detalii despre cum e implementat sitemul de fisiere.

6. 128 RAM

Sp de adrese pe 43 biti, pagini de 8kb, PTE (Page Table Entries) 8 bytes

Program: 10KB text+date incarcate la baza spatiului de adrese + 5KB stiva la capatul superior

(a) cat de mare ar fi o tabela liniara de pag pentru a mapa acest program in RAM?

- nr total de pagini in spatiul de adrese: 43 biti spatiu, dim pag = 8kb = 2^13 => nr de pagini = 2^(43-13) = 2^30

-nr total de pag \* dim unei intrari in tabela: 2^30\*8=2^33

=> dim unei tabele liniare pt a mapa acest program in RAM = 2^33 bytes

(b) mapare page table 3 niv: dim pag = 8KB = 2^13

43/13 39

Table of table of page table:

39 – 26

Table of page table:

25-13

Page table: 12-0

Offset: log2(2^13) = 13

(c) posibila implementare

1.Initializare: la pornire, sistemul ar aloca spatiu pentru tabela de pagini (intr o zona reervata sau prin alocare dinamica)

Dimensiunea tabelei: 2^30\*8 = 2^33 bytes

2.Incarcarea programului: se incarca cei 10kb de text si date si stiva de 5KB

3. Executarea programului

8. Aplicatie multimedia

3 threads: Tv – video, Ts Td – canalele stang si drept pt sursa stereo

(a) Se pot planifica aceste threaduri conform algoritmului Rate Monotonic? Com rez obtinut

Rate Monotonic: preemptive scheduling, taskul cu perioada cea mai scurta are prioritatea cea mai mare

Un set de n taskuri poate fi planificat daca indeplineste criteriul Liu si Layland:

Ci = timpul de calcul al taskului i

Ti= perioada taskului i

Tv: CTv = 50, TTv=100

Ts: CTs=2, TTs=10

Td:CTd=2, TTd=10

Suma devine: 50/100+2/10+2/10 = 0.9

n(2^(1/n)-1)=3(2^(1/3)-1)=0.78

0.9>0.78 => conform criteriului Liu si Layland setul de taskuri nu poate fi planificat folosind Rate Monotonic