Popa Marian Gabriel
431A

Protecția Memoriei

1. Introducere

Cele mai multe dintre sistemele de operare actuale implementează o forma sau alta de protecţie a memoriei pentru procesele utilizatorilor. Această protecţie face posibil accesul prin permisiune ceea ce determina daca o regiune de memorie alocata unui proces poate fi scris, citita sau executata de acel proces.[3]

Scopul principal al protecţiei memoriei este de a preveni procesele să acceseze zone de memorie care nu le aparţin. Acest lucru previne un bug dintr-un proces să afecteze alte procese, sau sistemul de operare în sine. [2]

[3] Tanenbaum, A. and Woodhull, A., Operating Systems: Design and Implementation, Pearson Prentice Hall, 3rd ed., 2006.
[2] Silberschatz, A., Galvin, P., and Gagne, G., Operating System Concepts, Wiley, 7th ed., 2004.

2. Originea problemei în legatură cu protecţia memoriei

In timp ce protecţia memoriei este o tehnica folositoare, care ajuta la îmbunătăţirea fiabilitaţii şi securitatea proceselor ea nu este foarte rafinata. Din acest motiv setarea permisiunii se poate aplica doar la paginile complete. Acest lucru limiteaza flexibilitatea, în special când sunt mici fragmente de memorie localizate una langa cealalta şi au nevoie de diferite tipuri de permisiuni.

Pentru solutionarea acestei probleme am ales tehnica Mondriana a protecţiei memoriei (**Mondrian Memory Protection**). MMP este o tehinca care extinde metodă tradiţională de protecţie a memoriei , în loc să seteze permisiunea la nivelul paginii, MMP suportă diferite permisiuni de acces pentru cuvinte individuale, însă este limititat tot la 2 biţi de permisiune. [4-5]

[4] Bishop, M., Computer Security: Art and Science, Addison-Wesley, 2003.
[5] Tsyrklevich, E. and Yee, B., "Dynamic detection and prevention of race conditions în file accesses," Usenix Security Symposium, 2003.

3. Managementul Memoriei[3]

In primul rând vom vorbi despre managementul memoriei la *Intel x86* pentru a putea compara cu modelul MMP

Sistemele de operare moderne impart spaţiul de adrese virtuale vizibil unui proces în sectiuni de dimensiuni egale, numite pagini. Fiecare pagina de memorie alocata unui program este reprezentata de o intrare în tabela de pagini în directorul cu pagini al programului(ierarhia paginilor în tabel). Această ierarhie este folosită de catre sistem şi procesorul managementului memoriei să mapeze memoria virtuală corespunzatoare cu frame-urile fizice din RAM. Această mapare este necesara pentru a gasi locatia adreselor virtuale în memoria fizică.[1]

Exemplu de transaltare a adresei virtuale intr-o adresa fizică:

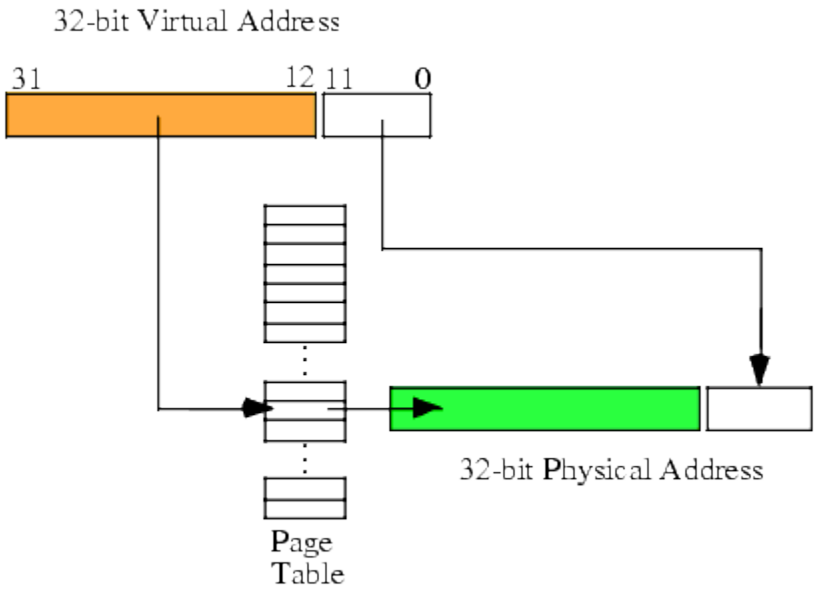

Figura 1:[link]

In afară de informaţia obţinută prin mapare , tebelul paginării conţine 2 biţi care informează CPU daca pagina este *read-only* sau este restricţionată la *supervisor* code.

*[1] **"Mondrian Memory Protection"**, Emmett Witchel, Josh Cates, and Krste Asanovic, Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X) , San Jose, CA, October 2002.*

[3] Tanenbaum, A. and Woodhull, A., Operating Systems: Design and Implementation, Pearson Prentice Hall, 3rd ed., 2006.

Similar cu arhitectura x86 MMP alocă 2 biţi pentru a stoca cele patru moduri diferite de acces pentru fiecare zona de memorie valabila în sistem:
->fara acces (no access)

->doar citire (read-only)

-citire-scriere (read-write)

-executare-citire (execute-read)

MMP foloseşte pentru a stoca acesti biţi intr-un tabel aditional numit *tabel de permisiuni.* Acest lucru permite sistemului să stocheze informaţia de protecţie mentru fiecare cuvant de memori . La fiecare acces catre o adresa de memorie, CPU caută bitii adresei de protecţie stocati în tabelul de permisiuni.

In ciuda flexibilitatii, MMP are un singur deficient, el nu poate asocia mai mult de 2 biţi de informaţie protejată unei zone de memorie. Pentru a rezolva această problema s-a incercat combinarea metodei protecţiei memoriei de la arhitetura Intel x86 cu modelul original Mondrian[5] şi s-a creat astfel modelul extins Mondrian. Acest model de arhitectura se imparte în 3 componente:

- ➔ Ierarhia protecţiei
- ➔ Controlul accesului la memorie

3.1 Ierarhia protecţiei[1]

Similar cu ierarhia paginilor în tabel folosită în arhitectura x86 pentru a face o mapare a adreselor virtuale în adrese fizice, MMP extins foloseşte 2 nivele ierarhice ale tabelei de protecţie. Fiecare tebela de protecţie are la randul ei intrari care indica catre paginile de protecţie. Fiecare cuvant alocat memoriei virtuale este reprezentat de o intrare în pagina de protecţie. Noul control introdus

CR6 serveste ca punct de intrare în ierarhia protecţiei.(figura 2)

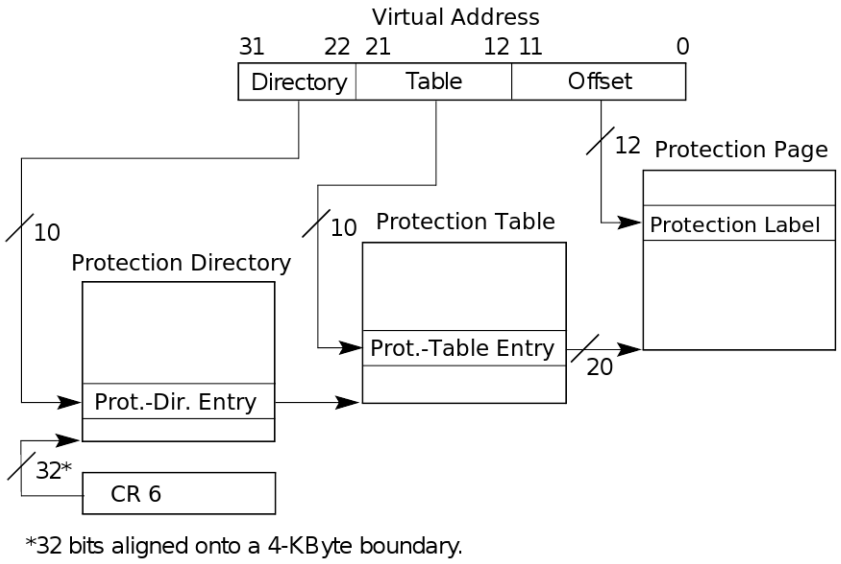

Figura 2: Ierarhia protecţiei. [1]

Pentru a salva spaţiu când cuvintele din tabelele de protecţie sunt identice, se folosesc 2 nivele diferite de granulare:

➔ Protecţie granulată ridicată: această metodă adaugă 30 de biţi de informaţie protejată fiecarui cuvant din spaţiul virtual de adrese.

➔ Protecţie granulată scăzută: această metodă stocheaza informaţia protejată direct în intrarea tabelului de protecţie, putand astfel să specificam 30 de biţi de informaţie protejată pentru o pagina virtuală de memorie completă.

Cu ajutorul mecanismelor *prot_mov, prot_and şi prot_or* utilizatorul poate modifica setarile protecţiei memoriei.

3.2 Controlul accesului la memorie[1]

Atunci când accesam memoria, CPU face o cautare a protecţiei informatiei pentru adresele corespunzatoare, acest lucru este facut prin verificarea ierarhiei protecţiei incepand cu valoarea curenta a registrului de contro CR6(figura 2).

Cand folosim protecţie granulată ridicată , intrarea în tabela de protecţie cautata de CPU serveste ca pointer catre pagina de protecţie, care conţine o eticheta de protecţie de 30-biţi folosită pentru contrulul memoriei de acces.

De indata ce toti cei 30 de biţi ai etichetei de memorie au fost receptionati, ei pot fi folositi pentru a decizia controlului la acces. Registrul de control CR5 stocheaza valoarea care determina controlul la acces al memoriei. Sunt prezenti şi doi biţi de acces numiti *read-mask şi write-mask.*

In cazul în care nu s-a gasit informaţie de protecţie accesul la memorie este garantat imediat.

Pentru a ajunge la decizia de control al acesului, sistemul ia cei 30-biţi de protecţie şi face operatia de AND logic cu bitii de acces (bit-mask depinzand daca operatia este de citire sau scriere). Rezultatul acestei operatii este un *protection token*. Similar voalorile rezultate din bitii stocati în registrul CR5 şi operatia de AND cu mask-ul respectiv obtinem un *control token*. Comparand *token-urile* se ia decizia daca accesul la memorie ar trebui acordat sau nu.

Exemplu de acordarea accesului la memorie:

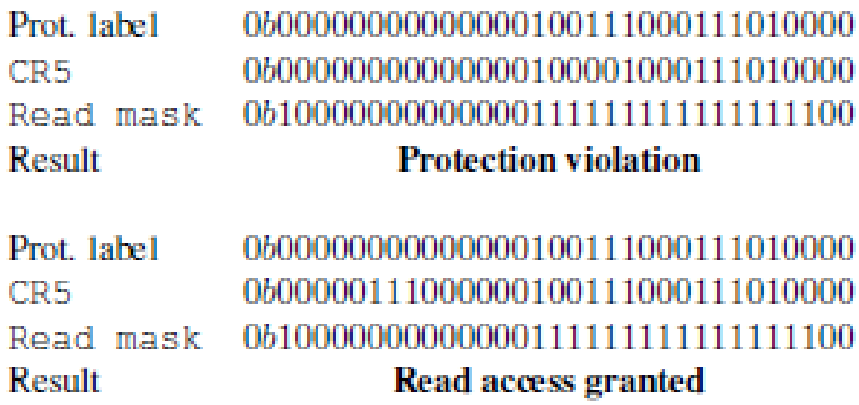

Figure 3: Access control decisions for a read access. [1]

4. Alte tehnici de protecţie a memoriei

➔ Protectia memoriei pentru sistemele Incorporate

- Prin folosirea unei metode simple de protecţiei a memoriei în dezvoltarea de sisteme incrorporate se măreşte fiabilitatea sistemelor fara a pierde din performante.

Protectia memoriei la sitemele incorporate

Intr-un model de procese, fiecarui proces ii este data o adresa virtuală completă, care este mapata, pagina cu pagina, şi corespund paginilor din memoria fizică. Adresele fizice sunt contruite de offsetul adreselor în asteptare din interiorul paginii rezultatand o adresa de 32-biţi (32-biţi arhitecture)

Fiecare proces poate face referiri decat la adrese din interiorul propriului spaţiu virtual, prevenind astfel accesul la memoria fizică care nu i-a fost acordata. Memoria fizică mapata se poate accesa conform permisiunii pentru acea pagina, asa cum este stocată în Unitatea de management a memoriei (MMU) de catre sistemul de operare la initializare(figura 4)

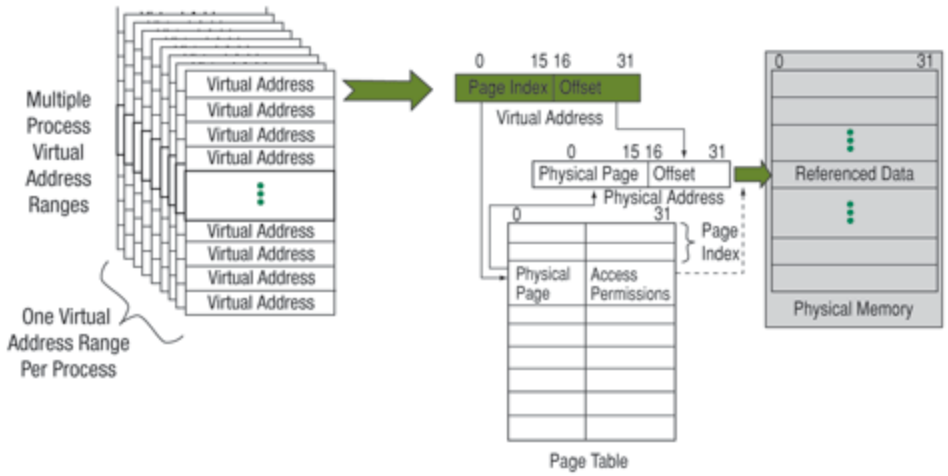

Figure 2 In a virtual memory system, addresses in each user's virtual address space are translated to reference code and data in physical memory.

Figura 4: [link]

Acest model ofera beneficii dezvoltatorului dâdu-i o masina virtuală fiecarui proces. Fiecare proces primeşte o adresa virtuală de 32-biţi care este mapata pentru memoria fizică pagină cu pagină. Acest lucru oferă protecţie mostenită intre procese.

Articol: [6] ->Eficient Memory Protection on Embeded Systems

Cuprins

Bibliografie Globală

[1] **"Mondrian Memory Protection"**, Emmett Witchel, Josh Cates, and Krste Asanovic, Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X) , San Jose, CA, October 2002.

[3] Tanenbaum, A. and Woodhull, A., Operating Systems: Design and Implementation, Pearson Prentice Hall, 3rd ed., 2006.

[2] Silberschatz, A., Galvin, P., and Gagne, G., Operating System Concepts, Wiley, 7th ed., 2004.

[4] Bishop, M., Computer Security: Art and Science, Addison-Wesley, 2003.

[5] Tsyrklevich, E. and Yee, B., “Dynamic detection and prevention of race conditions în file accesses,”
Usenix Security Symposium, 2003.

[6] Articol: Eficient Memory Protection on Embeded Systems