Legenda:

Cezar

Elena

Theo

Flv

Soro

11 iunie 2015

# 1. Ce cauzeaza trecerea din user mode in kernel mode? (7p)

Apelurile de sistem cauzeaza trecerea din user mode in kernel mode

# 2. Ce contine, in mod uzual, o intrare in tabela de pagini a unui proces? (7p)

Tabela de pagini a unui proces conține pointeri de pagini fizice. Indexul în tabelă este pagina virtuală. În general tabela de pagini mai conține și informații legate de permisiuni, validatate, dacă pagina a fost sau nu modificată. (Lucrari\_SO.docx)

# 3. De ce este, in general, mai rapida schimbarea de context intre 2 thread-uri ale aceluiasi proces decat schimbarea de context intre doua procese? (7p)

Surse de overhead pentru schimbarea de context între procese sunt schimbarea tabelei de pagini, care conduce la flush la TLB, algoritmul de alegere a următorului proces și schimbarea efectivă de context, cu salvarea registrelor procesului curent și restaurarea procesului ales. (lucrare 2 2014-2015 3CA var 1 ex 1).

# 4. Descrieti secventa de apeluri Linux prin care descriptorul 3 din tabela de descriptori de fisier a unui proces va referi iesirea standard, iar descriptorul 1 va referi fisierul a.txt (7p)

int fd;

dup2(1, 3);

fd = open(“a.text”, O\_CREAT | O\_WRONLY);

close(1);

dup2(fd, 1);

close(fd);

# 5. Ce se intampla in cazul operatiei up() pe un semafor? (7p)

Se incrementeaza valoarea semaforului. Daca exista cel putin un proces care asteapta semaforul acesta este deblocat de scheduler si valoarea semaforului este decrementata la loc. (<http://www.cyberiapc.com/os/blocking-semaphores.htm>)

# 6. Pe un sistem de fisiere un inode contine 7 pointeri directi catre blocuri de date si 2 pointeri cu indirectare simpla. Stiind ca un bloc are 4096 de octeti si ca un pointer ocupa 4 octeti, care este dimensiunea maxima a unui fisier pe acest sistem de fisiere? (10p)

Dim maxima a unui fisier = nr. maxim de blocuri intr-un inode \* dim bloc

Un pointer cu indirectare este un pointer catre un bloc care contine pointeri catre alte blocuri ce contin date => Nr blocuri pt un pointer cu indirectare = dim bloc / dim pointer

pointeri / bloc = dim bloc / dim pointer = 4096 / 4 = 1024

pointeri in blocurile indirecte = 2 \* 1024 = 2048

=> 2048 (de la blocurile indirecte) + 7 (de la blocurile directe) = 2055 blocuri totale

dim maxima a unui fisier = 2055 \* 4096 ~= 8MB

OSCE pg 559,560 - Combined scheme + schema pe 560

# 7. Intr-o zona a unui program exista urmatoarea secventa :

char arr[100];

[...] /\* instr oarecare nu afecteaza array-ul arr \*/

arr[23] = ‘a’;

arr[24] = ‘b’;

In ce situatie prima instructiune de atribuire NU cauzeaza page fault, dar a doua instructiune de atribuire cauzeaza page fault? (10p)

In cazul in care arr[23] se afla intr-o pagina incarcata in memoria fizica iar arr[24] intr-o pagina care inca nu e incarcata in memoria fizica. Pentru ca array-urile in C tin elementele in adrese consecutive arr[23] trebuie sa fie ultimul octet intr-o pagina iar arr[24] primul octet in pagina urmatoare. E corect. Am intrebat un guru :))

# 8. Un programator a scris un program in limbajul X care foloseste apelul recv(s, buf, 2000) iar dimensiunea buf este de 1000 de octeti. Sistemul in discutie foloseste tehnicile Address Space Layout Randomization (ASLR) si Data Execution Prevention (DEP). Poate fi acest bug exploatat? Daca da, explicati cum. (10p)

Un atacator poate ajunge să suprascrie o variabilă care este apoi folosită într-o comparație pentru a altera astfel fluxul de execuție. Dacă un atacator suprascrie o adresă de funcție (pointer de funcție sau adresa de retur) atunci acesta poate sări undeva în codul programului (codul existent al programului, nu o bibliotecă al cărei cod este randomizat - ASLR); dacă în acest cod se găsesc elemente care pot fi folosite de atacator (probabil da), acesta va altera execuția programului pentru a îl exploata.

(lucrare 3 2015-2016 3CA var 2 ex 1)

# 9. Dati exemplu de 3 mecanisme ale sistemului de operare care permit restrictionarea daunelor pe care le poate provoc un proces controlat de un atacator. (10p)

* separatia user mode si kernel mode
* memorie virtuala (pentru ca nu permite citirea saumodificarea memoriei altor procese)
* access control

# 10. Explicati cum stie sistemul de operare carui proces ii este destinat un segment TCP primit. Diferentiati intre segmnetele cu bit-ul SYn activat sau dezactivat. (10p)

Fiecare socket este identificat de adresa ip sursa, adresa ip destinatie si numerele porturilor sursa si destinatie. La primirea unui segment TCP sistemul de operare examineaza cele 2 adrese si porturi ale segmentului si face un lookup intr-un hash table cu conexiuni active pentru a determina procesul si socket-ul ce va primi pachetul. (OSCE pg 821 ultimul paragraf inainte de 18.11)

In cazul conexiunilor noi (clientul trimite un pachet cu bitul SYN activ) adresa destinatie si portul destinatie sunt cautate intr-un hashtable ce contine socket-uri ce asculta conexiuni noi. Socket-ul destinatie este notificat si cand procesul apeleaza accept pe acel socket se va crea noua intrare in hash table-ul de conexiuni active cu ambele adrese si porturi si aceasta va indica catre socket-ul returnat de accept si procesul care a apelat accept. (<http://www.tldp.org/LDP/tlk/net/net.html>)

# 11. Unui programator i se cere sa implementeze un server de imagini care va deservi un nr mare de utilizatori. Fiecare utilizator poate incarca sau descarca imagini de dimensiune mica (<1MB). Sistemul trebuie sa poata sustine simulatn minim 1000 de utilizatori (upload si download), iar timpul maxim de upload sau download trebuie sa fie 1s per fisiere. Sistemul trebuie sa retina pt fiecare utlizator nr de octeti incarcati sau descarcati, precum si nr total de octeti pt toti utilizatorii.

1. Ajutati programatorull sa aleaga hardware-ul minim necesar pt acest server. Alegeti placa de retea de viteza, dimensionati memoria RAM, alegeti nr de core-uri, procese, HDD vs SSD, etc. Explicati toate alegerile facute. (7p)
2. Specificati arhitectura serverului : daca vor fi folosite procese sau thread-uri (cate?), apeluri blocante sau neblocante, etc. (8p)
3. Pentru arhitectura aleasa, descrieti in pseudocod codul de tratare a unui client si modul in care statisticile sunt actualizate. (10p)

a. Placa retea: 1000 utilizatori \* 1MB/1s = 1000MB/s = 8 Gigabits / second => Trebuie sa foloseasca o placa de retea de 10Gbps.

Stocare: Pentru ca se pot face pana la 1000 de accese la zone random mici (<1MB) este necesar un mediu de stocare rapid cu latenta mica si capabil de multe IOPS (Input/Output Operations Per Second), deci un SSD (HDD are seek time prea mare si IOPS prea mic). Interfata folosita trebuie sa suporte o banda de 8 Gbps, deci SAS, PCI Express sau NVM nu SATA. Preferabil intr-o configuratie RAID pentru redundata.

Memorie: Consider ca fiecare transfer are nevoie de 1MB pentru a stoca fisierul in memorie, putem folosi zero-copy pentru a elimina copieri, plus alte informatii legate de transfer consider ca 5MB de memorie / transfer sunt suficienti. 5MB \* 1000 transferuri ~= 5GB memorie. Plus ce mai consuma sistemul de operare consider ca 8GB de memorie RAM sunt suficienti.

Procesor: Conexiunile de 10Gbps consuma mult cpu time, deci cel putin un Xeon quad core.

b. O abordare non-blocanta cred ca este cea mai buna pentru acest caz. Din cauza asta sunt suficiente atatea thread-uri cate poate hardware-ul sa duca concurent (de exemplu 8 pentru un quad core cu hyper threading). Acestea pot apartine unui singur proces (caz cel mai unreliable, daca un transfer cauzeaza un crash pica toate, dar cel mai performant) sau fiecare thread in propiul proces cu un proces parinte care sa preia transferuri si sa le trimita la procese (cel mai reliable, daca un transfer cauzeaza un crash pica doar o parte din transferuri, dar cel mai incet datorita contex switch-urilor intre procese). As opta pentru a 2-a varianta, mai reliable.

c.

parinte: //preia conexiuni noi, le paseaza la workeri si updateaza statisticile

int p = 0;

database\_update dbu;

while(e = wait\_event()) {

if (IsNewClient(e)) {

send\_to\_worker(worker[p], NewTransfer(e));

p = (p + 1) % num\_threads;

}

else if (IsTransferComplete(e)) {

memset(dbu, 0);

dbu.transfer\_size = e.transfer\_size;

if (e.is\_upload)

dbu.upload\_size = e.transfer\_size;

else

dbu.download\_size = e.transfer\_size;

database.send\_update(dbu);

}

}

worker: //preia transferuri de la parinte si il notifica cand acestea sunt finalizate

while(e = wait\_event()) {

if (IsNewTransfer(e)) {

int fd = open(get\_file\_path(e));

if (e.upload) {

sendfile(fd, e.socket, e.transfer\_size);

}

else {

sendfile(e.socket, fd, e.transfer\_size);

}

}

else if(IsTransferComplete(e)) {

send\_to\_parent(e);

}

}

15 iunie 2015

# 1. De ce procesele I/O intensive primesc, in general, o prioritate mai mare decat procesele CPU intensive? (7p)

Preferăm o cuantă de timp mai mare proceselor I/O intensive pentru că acestea au probabilitate mare de a se bloca și de a elibera astfel procesorul, lăsând locul altor procese. Un proces I/O intensiv va părăsi astfel mai rar procesorul din cauză exprirării cuantei, mărind productivitatea (throughput-ul) sistemului (o schimbare de context suplimentară cauzează overhead). (lucrare 2 2015-2016 3CC var 1 ex 1).

# 2. De ce consideram apelul chroot() o forma de sandboxing? (7p)

Deoarece poate izola procesele care ruleaza in chroot de restul sistemului de fisiere (curs 13 pg 30)

# 3. Ce se intampla in cazul operatiei down() pe un semafor? (7p)

Se decrementeaza valoarea semaforului daca aceasta este pozitiva. Altfel threadul care face apelul se va bloca pana cand valoarea semaforului va deveni pozitiva (adica se va apela up din alt thread) si va decrementa imediat valoarea. Doar un singur thread este deblocat cand un thread apeleaza up pe semafor. (<http://www.cyberiapc.com/os/blocking-semaphores.htm>)

# 4. De ce schimbarea de context intre doua user level threads ale aceluiasi proces este mai rapida decat schimbarea de context intre doua kernel level threads ale aceluiasi proces? (7p)

Thread-urile user level sunt mult mai simplu implementate si nu interactioneaza cu scheduler-ul sistemului de operare.

<http://cs.stackexchange.com/a/1066>

# 5. Ce contine o intrare in TLB (Translation Lookaside Buffer)? (7p)

O cheie (numarul paginii) si o valoare (numarul frame-ului in care este stocata pagina). OSCE pg 373

# 6. Pe un sistem cu suport DEP (Data Execution Prevention), un atacator urmareste sa realizeze un apel mprotect(..., PROT\_EXEC, …) pe un set de pagini apartinand stivei. Ce urmareste atacatorul cu un astfel de apel? Cum il va ajuta in continuarea atacului? (10p)

(nu sunt foarte sigur) Urmareste sa schimbe drepturile unei zone de memorie writeable in executable. Probabil ca atacatorul a descoperit o vulnerabilitate de tip buffer overflow si doreste sa o foloseasca pentru a injecta si executa cod folosind un atac de tipul return to libc. Daca reuseste sa faca acest apel DEP nu va proteja procesul deoarece protectia paginii se schimba din writeable in executable, nu in writeable + executable care este impedicat de DEP.

# 7. Un programator masoara durata celor doua instructiuni de mai jos, realizate consecutiv:

p = mmap(NULL, PAGE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

p[0] = ‘a’;

In urma masuratorii observa ca prima instructiune (mmap) dureaza 900 ns iar a doua dureaza 3000 ns. Ce putem spune despre overhead-ul de timp cauzat de un apel de sistem fata de overhead-ul de timp cauzat de page fault handler? (10p)

Un page fault este mult mai costisitor decat un apel de sistem deoarece page fault-ul este rezolvat de un handler care ruleaza in kernel mode, deci ambele cauzeaza un context switch. In plus handler-ul trebuie sa initializeze pagina sau s-o incarce din swap si sa updateze TLB-ul.

# 8. Intr-un program se realizeaza apelul (10p):

fd = open(“/home/student/myfile”, O\_RDONLY);

Apelul reuseste. Stim ca /home/student/myfile este o legatura simbolica. Cate inode-uri sunt citite in cadrul apelului open() de mai sus?

4 (pentru /, home, student, myfile) pentru a obtine calea din legatura simbolica + inodurile citite pentru rezolvarea legaturii simbolice

(<https://www.quora.com/Linux-Kernel-How-do-the-path-look-up-mechanism-namei-work-in-Linux>)

# 9. De ce, in general, pornirea unui proces dureaza mai mult la prima sa pornire decat la urmatoarele porniri? (10p)

Din cauza page cache-ului - Datele accesate recent de pe disc/sistemul de fișiere sunt reținute în memorie pentru acces rapid. Șansele sunt mari ca acele date să fie reaccesate în viitor. Memoria fiind mult mai rapidă ca discul, se mărește viteza de lucru a sistemului.

(lucrare 4 2013-2014 3CA var 1 ex 1)

# 10. In ce situatie atat apelul send() pe transmitator, cat si apelul recv() pe receptor (in aceeasi conexiune TCP) sunt simultan blocate? (10p)

In cazul in care transmitatorul are bufferul de send full si receptorul are bufferul de receive gol. Apelurile se vor debloca cand kernelul transmitatorului trimite cu succes o parte din continutul bufferului de send la receptor si primeste ACK.

# 11. Se propune implementarea unui profiler de aplicatii multithreading cu obiectivul de a obtine informatii legate de profilul de performanta al acestor aplicatii. (25p)

a. Ce informatii/metrici sunt utile de a fi furnizate de profiler dezvoltatorului aplicatiei?

In ce forma sunt utile sa fie furnizate acestea dezvoltatorului (numere, medii, grafice, tabele)? (7 puncte)

b. Ce facilitati trebuie sa ofere sistemul de operare si hardware-ul pentru functionarea profile- rului? (6 puncte)

c. Cum va accesa profilerul facilitatile oferite de sistemul de operare si hardware (apeluri de sistem, memorie partajata, sisteme de fisiere, dispozitive virtuale etc.)? (5 puncte)

d. Ce facilitati/metrici vor fi de interes pentru dezvoltatorul unei aplicatii de transcoding

(conversie video)? Cum va fi folosit profilerul de catre dezvoltator pentru a profila/optimiza

aplicatia de transcoding? (7 puncte)