Sisteme de operare

10 septembrie 2009

Timp de lucru: 70 de minute

NOTĂ: toate răspunsurile trebuie justificate

1. Care din următoarele acțiuni consumă cel mai mult timp în cazul unei schimbări de context între două thread-uri ale aceluiași proces:

- schimbarea registrelor
- flush TLB
- schimbarea tabelei de pagini
- schimbarea tabelei de descriptori de fișier

În cazul schimbării de context între două thread-uri ale aceluiași proces nu se face flush la TLB, nu se schimbă tabela de pagini, nu se schimbă tabela de descriptori de fișier. În consecință, deși foarte rapidă, acțiunea care consumă cel mai mult timp este schimbarea registrelor.

2. Un sistem dispune de magistrală de date și registre pe 32 de biți (sizeof(unsigned long) = 32). Sistemul folosește paginare simplă (non-ierarhică) și nu are memorie cache, nici TLB. Știind că un acces la memorie durează 50ns iar o pagină este de 4KB, cat va dura secvența de mai jos? Se presupune că vectorul buffer este alocat în RAM (memoria fizică) și că valoarea contorului *i* se păstrează într-un registru (nu folosește memoria).

```
unsigned long buffer[32*1024];

for (i = 0; i < 32 * 1024; i++)
        buffer[i] = i;
```

*În absența TLB fiecare acces la memoria fizică necesită accesarea tabelei de pagini (aflată tot în memoria fizică). Astfel, pentru fiecare dintre cele 32*1024 de accese la buffer vor exista încă 32*1024 accese la tabele de pagini. Rezultă 64*1024 accese la memorie cu o durată totală de 64*1024*50ns.*

3. Care dintre variantele chroot, respectiv OpenVZ oferă un grad mai mare de securitate?

chroot oferă "încapsulare" (securizare) doar la nivelul sistemului de fișiere. OpenVZ oferă securizare la nivelul proceselor, memoriei, procesorului, rețelei, utilizatorilor etc. OpenVZ oferă, așadar un grad mai mare de securitate.

4. În cadrul unui proces cu mai multe thread-uri, un thread execută următoarea secvență de cod (pseudo C):

```
int esp;
int stack_val;

/* se obtine valorea registrului esp (registru de stiva) al thread-ului planificat anterior */
esp = get_former_esp();
stack_val = *(esp + 4);
```

Care va fi rezultatul execuției secvenței de mai sus pe un sistem în care stiva crește în jos?

Întrucât stiva crește în jos, valoarea esp+4 va puncta către o zonă alocată din stiva fostului thread. Execuția de mai jos va rezulta în obținerea acelei valori (nu se va obtine segmentation fault decât dacă thread-ul anterior și-a încheiat execuția).

5. Pe un sistem de fișiere MINIX se execută operația:

```
fd = open("a.txt", O_RDWR | O_CREAT | O_TRUNC, 0644);
```

Precizați ce se întâmplă la nivelul sistemului de fișiere (inode, inode bitmap, zone bitmap, data block, dentry etc.) în cazul în care fișierul există sau nu există pe disc.

Dacă fișierul există se găsește dentry-ul acestuia și se obține inode-ul aferent și se citește inode-ul în memorie.
Dacă fișierul nu există se creează un inode nou. Pentru aceastase parcurge bitmapul de inode-uri și se alocă un inode. Se completează cu 1 poziția liberă găsită. Se creează un dentry cu numele "a.txt".
Dacă fișierul exista este trunchiat. Se parcurg pointer-ii de blocuri ai inode-ului și se marchează cu NULL (sau ceva echivalent). Se parcurge bitmapul de blocuri și marchează cu 0 pozițiile aferente acelor blocuri. Dacă fișierul avea mai mult de 7 blocuri se citește și completează cu NULL blocul pentru dereferențieri simple.

6. Precizați o soluție de sincronizare pentru problema formării moleculei de oxid de fier (Fe_2O_3) (ca alternativă la problema formării apei).

```
void fe_fun()
{
        m.enter();
        fe_count++;
        if (o_count >= 3) {
                if (fe_count < 2)
                        m.fe_cond.wait();
                else {
                        m.o_cond.signal();
                        m.o_cond.signal();
                        m.o_cond.signal();
                        m.fe_cond.signal();
                        fe_count -= 2;
                }
        }
        else
                m.fe_cond.wait();
        m.leave();
}
```

```
void o_fun()
{
        m.enter();
        o_count++;
        if (o_count >= 3 && fe_count >= 2) {
                m.o_cond.signal();
                m.o_cond.signal();
                m.fe_cond.signal();
                m.fe_cond.signal();
                if (o_count > 3) {
                        m.o_cond.signal();
                        m.o_cond.wait();
                }
        }
        else
                m.o_cond.wait();
        m.leave();
}
```

7. Biblioteca standard C oferă programatorului funcția calloc (alocare cu zeroing). De ce este nevoie și de oferirea funcției malloc?

Funcția malloc este mai rapidă – nu face zeroing și permite demand paging.

8. Pe un sistem care dispune de 3 pagini fizice (frames) și folosește un algoritm de înlocuire a paginii de tip NRU se execută următoarea secvență:

```
1r, 2w, 4r, 1w, 3r, 2w, 1w, 4w, 3r, 3w, 1r, 2r, 5r, 2w, 6r, 3w, 1w, 2r
```

Câte page fault-uri au loc? 1r înseamnă operație de citire în cadrul paginii virtuale 1; 2w înseamnă operație de scriere în cadrul paginii virtuale 2.

Conținutul celor 3 pagini fizice, împreună cu evenimentul aferent este prezentat, evolutiv, în tabelul de mai jos; la fiecare două pagefault-uri se resetează bitul referenced):

frame 1	1r (R)	1r (R)	1r	1w (W)	3r (R)	3r (R)	3r	3r (R)	3w (RW)	3w (W)	2r (R)	2r (R)	2w (W)	6r (R)	6r (R)	6r	2r (R)
frame 2		2w (W)	2w (W)	2w (W)	2w (W)	2w (W)	4w (W)	4w (W)	4w (W)	4w (W)	4w (W)	5r (R)	5r (R)	5r	3w (W)	3w (W)	3w (W)
frame 3			4r (R)	4r (R)	4r	1w (W)	1w (W)	1w (W)	1w (W)	1w (RW)	1w (RW)	1w (W)	1w (W)	1w (W)	1w (W)	1w (W)	1w (W)
	PF	PF	PF	PF	PF	PF	PF		PF		PF	PF	PF	PF	PF		PF

9. Fie secvența de program de mai jos:

```
int main(void)
{
        char *a;
        int i;

        for (i = 0; i < 10; i++)
                a = malloc(1);

        return 0;
}
```

La rulare se observă (prin folosirea unui profiler) că primul apel malloc durează semnificativ mai mult decât celelalte 9. Care este motivul? Toate apelurile reușesc (întorc o adresă validă) și nu există nici o modificare adusă apelului malloc.

Primul apel malloc generează un page fault, urmarea fiind alocarea unei pagini fizice întregi (chiar dacă se solicită alocarea unui singur octet). Următoarele apeluri vor aloca octeți din cadrul aceleiași pagini – nu mai este generat un page fault și nu se aloca alte pagini.

10. Are sens folosirea operațiilor de tipul Overalapped I/O pe un sistem care dispune de un singur hard-disk?

Da. Operațiile overlapped I/O permit o planificare mai eficientă a operațiilor de I/O la nivelul nucleului și permit aplicației să ruleze

11. Descrieți o situație în care două procese partajează o pagină virtuală (din spațiul virtual de adrese).

Fiecare proces are propriul spațiu de adrese. Nu există noțiunea de partajare a unei pagini virtuale.