Nume și grupă:

Sisteme de Operare

9 iunie 2011

Timp de lucru: 60 de minute

Notă: Toate răspunsurile trebuie justificate

1. Cum se modifică zona de cod (text) a unui proces în cazul creării unui nou thread cu implementare la nivelul nucleului (kernel-level threads)?

2. Se realizează un apel `mmap` pe următoarele platforme:

- un container OpenVZ
- o mașină virtuală Xen
- un sistem emulat prin Bochs (emulator)

Care este ordinea operațiilor în funcție de timpul de rulare (de la mic la mare)? Justificați.

3. Justificați valoarea de adevăr a afirmației: *În cazul unui symlink, pointerii din cadrul inode-ului inițial (fișierul origine) și pointerii din cadrul inode-ului symlink-ului referă aceleași blocuri.*

4. Cum poate un apel `printf` să conducă la scrierea informației pe un socket?

5. Un semafor este inițializat la valoarea 5. Mai multe thread-uri execută următoarea secvență de pseudocod:

```
down(&sem);
/* critical section */
up (&sem);
```

Care este numărul maxim de thread-uri care poate aștepta, la un moment dat, la semafor, respectiv care se pot găsi, la un moment dat, în regiunea critică (critical section)?

6. De ce paginarea (în cazul memoriei) previne fragmentarea externă, dar nu și fragmentarea internă?

7. Fie următoarea secvență de cod:

```
int flag = 0;

void  *func(void *arg)
{
        int a;

        if (flag == 0) {
                flag = 1;
                a = 5;
        }
        else {
                a++;
        }
        printf(\a= %d\n", a);

        return NULL;
```

}

Funcția `func` este executată, serial/succesiv (nu se întrepătrund), de două thread-uri. Ce informații vor fi afișate?

8. Care dintre următoarele operații cauzează un TLB flush:

- write blocant apelat de un proces singlethreaded
- write blocant apelat de un proces multithreaded cu implementare în user-space
- write blocant apelat de un proces multithreaded cu implementare în kernel-space

9. În urma rulării unui executabil folosind `strace` (pentru analiza apelurilor de sistem) rezultă următoarele apeluri legate de biblioteca standard C (`libc.so`):

```
open("/lib/libc.so.6", O_RDONLY)            = 3
mmap(NULL, 3680360, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f312ab35000
mmap(0x7f312aeae000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x179000)
= 0x7f312aeae000
```

De ce primul apel mmap folosește `PROT_READ|PROT_EXEC`, iar al doilea folosește `PROT_READ|PROT_WRITE`?

10. Fie următoarea secvență de cod:

```
char *a;

void func(void)
{
    for (int i = 0; i < NUM_PAGES; i++)
        a[i*PAGE_SIZE] = 42;
}

int main(void)
{
    /* pseudocod */
     a = mmap(NUM_PAGES * PAGE_SIZE);
    for (int i = 0; i < NUM_PAGES; i++)
        a[i*PAGE_SIZE] = 42;

     /* pseudocod */
     create_thread(func);
     wait_thread();

     return 0;
}
```

Câte page fault-uri se obțin în cadrul funcției `func`?

> În conformitate cu ghidul de etică al Catedrei de Calculatoare, declar că nu am copiat și nu voi copia la această lucrare. De asemenea, nu am ajutat și nu voi ajuta pe nimeni să copieze la această lucrare.
>
> **Nume și grupă:**
>
> **Semnătură:.....**