Nume și grupă:

Sisteme de Operare

8 iunie 2014

Timp de lucru: 60 de minute

Notă: Toate răspunsurile trebuie justificate

1. (**7 puncte**) Un apel `mmap()` rezervă 16 pagini de memorie virtuală. Câte pagini de memorie fizică alocă apelul?

2. (**7 puncte**) De ce este dezavantajoasă folosirea unei cuante de timp prea mari pentru planificarea proceselor?

3. (**7 puncte**) Cu ce diferă un spinlock de un mutex?

4. (**7 puncte**) Dați exemplu de apel de sistem care poate conduce la o schimbare de context între procese. Explicați inclusiv în ce condiții se produce schimbarea de context.

5. (**7 puncte**) Apelurile `accept()` și `recv()` au o sintaxă de forma:

```
accept(sockfd1, ...)
recv(sockfd2, ...)
```

unde `sockfd1` și `sockfd2` sunt descriptori de socket. Cu ce diferă cei doi socketi?

6. (**10 puncte**) Într-un executabil sunt definite secțiunile `text` (codul programului), `data` (variabile globale) și `rodata` (variabile read-only). Care secțiuni vor fi partajate de două procese pornite separat din acest executabil?

7. (**10 puncte**) Ignorând câmpurile de tip timestamp din cadrul unui inode, dați un exemplu de apel de sistem de lucru cu fișiere (din forma `open()`, `read()`, `write()`, `seek()`, `close()`, `chmod()`, `stat()` etc.) care modifică un inode și altul care nu modifică un inode.

8. (**10 puncte**) Un sistem are suport DEP (*Data Execution Prevention*) dar nu are suport ASLR (*Address Space Layout Randomization*). Precizați cum se face un atac de tipul *return-to-libc*. Cum se obține adresa/adresele necesare?

9. (**10 punct**) Fie secvența de instrucțiuni de mai jos:

```
printf(\%d\n", *a);
printf(\%d\n", *(a+1));
```

unde `a` este un pointer la un întreg (`int *`). Dați exemplu de situație în care prima instrucțiune **NU** cauzează page fault, dar a doua cauzează page fault.

10. (**10 puncte**) Care este un avantaj, respectiv un dezavantaj al folosirii suportului de *huge pages*? Adică pagini de 2MB (*2 megabytes*) în locul paginilor de 4KB (*4 kilobytes*).

11. (**15 puncte**) Dorim să implementăm un alocator îmbunătățit de memorie. Alocatorul va expune funcțiile `malloc()`, `calloc()`, `realloc()` și `free()`, apeluri standard în lucrul cu memoria. În back end va folosi apelurile de sistem de tip `mmap()` sau `brk()` expuse de sistemul de operare pentru rezervarea de memorie virtuală. Cerințele alocatorului sunt viteză foarte bună și thread safety.

Ce structuri interne veți folosi în cadrul alocatorului pentru gestiunea alocărilor?

Ce probleme posibile (de viteză/eficiență) pot apărea la nivelul alocatorului?

Cum veți asigura viteză bună de alocare/dezalocare?

Ce fel de aplicații/scenarii de test veți folosi pentru a testa alocatorul?

În conformitate cu ghidul de etică al Departamentului 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:.....