# Examen la Sisteme de operare 2
# - probleme -

31.05.2004

1. [2.5 puncte] Să se implementeze apelurile sistem necesare unui mecanism simplu de comunicare între procese prin *pipe*:

   - `int pipe_open();`
     Crează un *pipe* şi întoarce un identificator (similar unui *file descriptor*).
   - `int pipe_write(int pipe, void *data, int len);`
     Scrie (în `pipe`) `len` octeţi de la adresa `data`. Funcţia întoarce numărul de octeţi scrişi.
   - `int pipe_read(int pipe, void *data, int len);`
     Citeşte (din `pipe`) octeţi la adresa `data`. Datele citite se scot din *pipe*. Funcţia întoarce numărul de octeţi citiţi, cel mult `len`.
   - `int pipe_close();`
     Închide *pipe*-ul. Orice scriere ulterioară în acest *pipe* va eşua. O citire din acest *pipe* va citi datele ce se mai află în pipe, apoi va intoarce valoarea 0.

   Toate funcţiile întorc *-1* în cazul apariţiei unei erori. Funcţia `pipe_open` întoarce un *identificator* pozitiv care desemnează semaforul utilizat în celelalte operaţii.

   Toate operaţiile de mai sus vor fi scrise astfel încât să fie atomice.

   Se poate considera că există un număr maxim oarecare posibil de *pipe*-uri. Numărul de octeţi maxim ce pot fi în *pipe* (pot fi scrişi fără a fi citiţi) la un moment dat este 1024.

   Macro-ul `current` indică PID-ul procesului curent. Următoarele funcţii se consideră implementate la nivelul nucleului şi pot fi folosite pentru rezolvarea problemei:

   - `void lock(lock_t the_lock);`
   - `void unlock(lock_t the_lock);`
   - `void initlock(lock_t the_lock);`
   - `void enable_intr();`
   - `void disable_intr();`
   - `int copy_to_user(void *from, void *to, int length);`

- `int copy_from_user(void *from, void *to, int length);`
- `void schedule(void);`
- `void *kmalloc(int size);`

Notă: nu este necesar să utilizaţi *toate* aceste funcţii.

De asemenea, sistemul posedă o tabelă de procese ca cea de mai jos:

```
struct task {
int state; /* READY, RUNNING or BLOCKED */
...
} ptable[500];
```

Dacă este necesar, puteţi adăuga câmpuri unei intrări în tabelă.

2. [0.75 puncte] Procese dintr-un sistem folosesc semafoare. Aceste semafoare sunt semafoare pot lua valorile 0 şi 1. Procesele din acest sistem pot avea stările READY, RUNNING şi BLOCKED. Vom nota cu S1, S2, ..., semafoarele. Asupra semafoarelor se pot efectua operaţiile `down()`, care decrementează semaforul şi `up()`, care incrementează semaforul.

   În continuare vom da trei secvenţe în care se precizează starea unui proces la un moment dat, o operaţie pe care o efectuează acel proces la momentul respectiv şi starea în care va trece procesul în urma operaţiei efectuate.

   Precizaţi, de fiecare dată, dacă situaţia este posibilă (sau de ce nu este posibilă) şi valoarea semaforului corespunzător înainte şi după operaţie, acolo unde este posibil.

   (a) [READY] S1`.down()` - [BLOCKED]
   (b) [RUNNING] S2`.down()` - [RUNNING]
   (c) [RUNNING] S3`.up()` - [BLOCKED]

   Notă: de exemplu pentru (a), se va citi: procesul este în starea READY şi efectuează operaţia `down()` pe semaforul S1. În urma acestei operaţii el trece în starea BLOCKED.

3. [0.75 puncte] Un proces are primele intrări în tabela de pagini după cum urmează:

| Virt. page | Phys. address | Valid |
|---|---|---|
| 0 | 12288(0x3000) | yes |
| 1 | 00000(0x0000) | yes |
| 2 | 20480(0x5000) | yes |
| 3 | 30222(0x760E) | no |
| 4 | 08192(0x2000) | yes |

   Ştiind că sistemul foloseşte doar paginare, cu dimensiunea paginii de 2kB şi cea a adresei de 32 biţi, să se precizeze:

   - Numărul de pagini virtuale pe care le poate adresa un proces
   - La ce adresa fizică se găseşte octetul cu adresa 9555 (0x2553)
   - La ce adresa fizică se găseşte octetul cu adresa 7055 (0x1B8F)