Pb. cina 5 filosofi

```
#define N 5
#define LEFT (i+N-1)%N)
#define RIGHT (i+1)%N)
#define THINKING 0
#define HUNGRY 1
#define EATING 2
typedef int semaphore;
int state[N];
semaphore mutex = 1;
semaphore s[N];
void philosopher (int i)
{   while (TRUE)
    { think();
      take_forks(i);
      eat();
      put_forks(i); }
}

void take_forks (int i)
{ down(mutex);
  state[i] = HUNGRY;
  test(i);
  up(mutex);
  down(s[i]); // asteptare pt ca var. globale int o.
}
void put_forks (int i)
{ down(mutex);
  state[i] = THINKING;
  test(LEFT);
  test(RIGHT);
  up(mutex);
}
```

```
void test (int i)
{ if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
      up(s[i]);
}
```

3. Problema cititori - scriitori

4. Problema frizerului somnoros

o sală de așteptare, un client nu poate aștepta în picioare

```
#define CHAIRS 5
typedef int semaphore;
semaphore customers = 0;
semaphore barbers = 0;
semaphore mutex = 1;
int waiting = 0; // nr de clienți în așteptare

void barber()
{  while(TRUE)
   {  down(customers); // doarme dacă nu are clienți
      down(mutex);
      waiting--;
      up(barbers);
      up(mutex);
      cut_hair();
   }
}
void customer()
{ down(mutex);
  if(waiting < CHAIRS)
  { waiting++;
    up(customers); // trezește frizerul
    up(mutex);
    down(barbers);
    get_haircut();
  } else up(mutex);
}
```

Planificarea proceselor (scheduling)

FIFO → NU

- priorități

ROUNA ROBIN (principiul round-robin) → împarte procesele în clase de priorități

$ nice myExe // $ ./myExe → seteaza prioritatea procesului

$ nice -nr myExe // nr = nr de unități cu care scad prioritatea

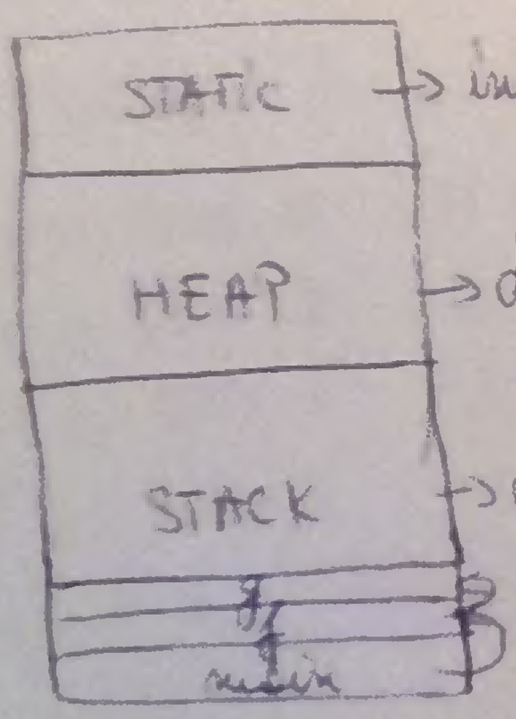
Doar root poate creste prioritatea proceselor.

- preemptiva (a p-a) : la a doua rulare îi acordă o prioritate în fct. de timpul și resursele consumate la prima lansare (primele)

Securitatea și gestiunea memoriei

Securitatea : un proces are propria zonă de mem. și nu poate scrie/citi în/din altă parte

Zona de date



→ init. cu 0 : var globale + var. locale statice

→ date alocate dinamic în tp execuției (ex malloc, new

→ arg + var locale ale fct.

void *malloc (int dim) → NU init

void *calloc (int nr, int dim) → init. zona cu 0.

nr elem. dim. elem. pt. vectori

calloc(n, m) ⟺ malloc (n * m).

void *realloc (void *p, int dim)

↓ dacă are spațiu memorie vecină

dacă nu există spațiu liber de 200, copiază 100 + adaugă 100 altfel la altă adr.

void free (void *p)

Pericol: - prin apeluri incorecte pot strica harta mem.

realloc si free se fac doar pe pointeri obt. Cu malloc sau calloc. si exact pointerul respectiv (nu increm sau decr.) sau NULL.

- malloc (100 * sizeof(int)) -> aloca 0-99.
  se scrie la 100, dar nu da eroare -> se strica mem.