Temă: de stabilit examenele pt data viitoare

- `int *x = NULL;`
  `*x = 55;   // sigsegv (segmentation fault)`

- setjmp.h

- sigjmp_buf – memorează un context de proces.
  apel: int sigsetjmp(sigjmp_buf context, int opțiuni)
  (context ↓ pointer; opțiuni = 0)

codul de retur: – inițial este 0

! să nu folosim var globale la examen

În handler o să executăm un apel la
int siglongjump(sigjmp_buf context, int val)
(context → var. globală)
↳ ne trimite înapoi în punctul ⊛ — codul de retur al lui sigsetjump

```
⊛ sigsetjmp(context, 0);

int rc;
rc = sigsetjump(context, 0);
switch (rc)
{
   case 0:
       break;
   case 1:
      /* SIGSEGV */
   case 2:
      /* SIGILL */
   case 3:
      /* SIGBUS */
}
/* Afișare meniu */
```

Fire de execuție (thread)

pthread – bibliotecă unix

! examen → ° Deadlock – concurență între procese

Schemă Deadlock direct → Avem două procese și două resurse, P_1 și P_2, respectiv, R_1 și R_2.

$P_1 \leftarrow R_1$

$P_2 \leftarrow R_2$

$P_1 \leftarrow R_2$ /* Așteaptă după P_2 */

$P_2 \leftarrow R_1$ /* Așteaptă după P_1 */ > procesele sunt în stare sleeping, în așteptare

S.O Alege unul dintre procese ca victimă (iese cu cod de retur -1, eroare: EDEADLCK) – resursele lui sunt **eliberate**

Criterii de alegere: - nr funcții ale resurse consumate
- ——— timp consumat

Îl alegem ca victimă pe cel ca "costul" cel mai mic

Deadlock – poate fi direct sau indirect

Ex: indirect

$P_1 \leftarrow P_2 \leftarrow P_3 \leftarrow P_4 \leftarrow \dots \leftarrow P_n \leftarrow P_1$

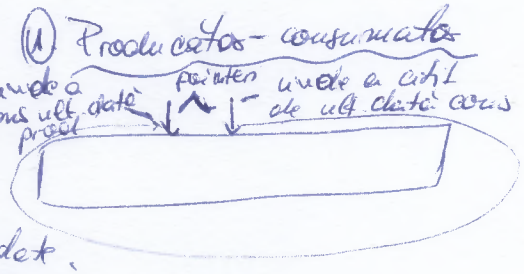
direct: - caz particular pt $n=2$.

Metode de prevenire

- dacă procesul nu mai are nevoie de resursă să o elibereze imediat

° Probleme clasice de concurență

(1) Producător - consumator



buffer circular

prod - produce date.

cons - consumă una după cealaltă datele

Prod - tb să fie înaintea cons.

Prod. - ast. în situația de buffer plin

Cons. - ast. în situația de buffer gol

Ex 5

```
/* sol greșită */
# define N 100
int count = 0; // câte elem. s-au produs și încă în buffer
void producer()
{
    int item;
    while (true)
        {
            item = produce_item();
            if (count == N) // buffer plin
                sleep();
            insert_item(item); // scriu în buffer
            count ++;
            if (count == 1)
                wakeup(consumer);
        }
}

void consumer()
{
    int item
    while (true)
        {
            if (count == 0)
                sleep();
            item = remove_item(item); // scot din buffer
            count --;
            if (count == N-1)
            wakeup(producer);
        consume_item(item);
}
```

```
/* sol corectă - cu semafoare */
# define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
```

```
void producer()
{
    int item;
    while (true)
        {
            item = produce_item();
            down (empty); // "am pus in buffer"
            down (mutex); // mutex arbitrează zona critică: val <0, 1
            insertitem (item);
            up (mutex);
            up (full);
        }
}

void consumer()
{
    int item;
    while (true)
        {
            down (full);
            down (mutex);
            item = remove_item();
            up (mutex);
            up (empty);
        }
}
```

- se poate seria cu cozi de mesaje - fără ipc nowait pt < msgrcv, msgsnd

⑫ Problema celor 5 filozofi