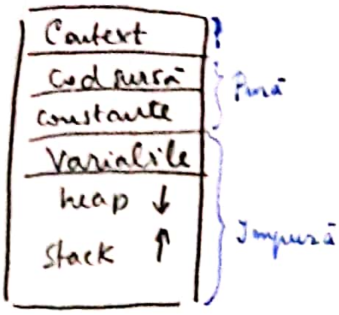
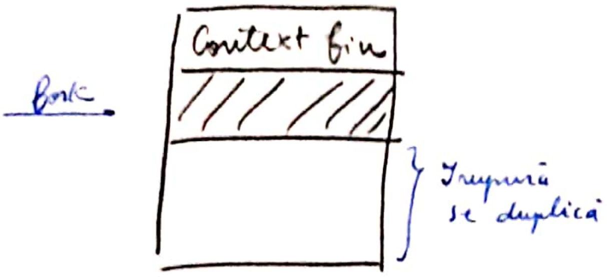
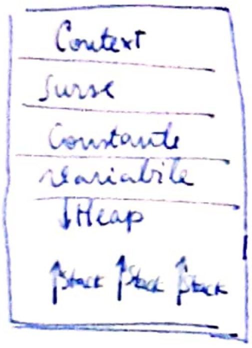
18.04.2018

Curs 8 - ~~OS~~ Sisteme de operare



fork





→ un stack pentru fiecare thread

→ signatura unei funcții pt. thread

```
void *f (void*a) {
    pthread-mutex-lock (&mtx);
    int i;
    for (i=0; i<10000; i++){
        a++;
    } pthread_mutex_unlock (&mtx);
    return NULL;
}
```

cea mai restrânsă zonă critică

← se va executa doar de un thread odată

```
gcc -Wall -g -o th-1 th-1.c -lpthread
```

libpthread.a → STATIC LIB
libpthread.so → DINAMIC LIB

mutex - mutual exclusion

---

thread-urile au handle

```
pthread_create (&t[i], NULL, f, NULL)
pthread_t t [100]; → handle-uri
```

→ caracteristici

→ arg pt. funcții

→ aici când creem threaduri, ele și pornesc

```
pthread_join (t[i], NULL); //așteptăm după theaduri
// dacă nu așteptăm după thread-~~uri~~uri, comportamentul
// este nedefinit.
```

→ putem primi ce returnează threadul

```
pthread_mutex_init (&mtx, NULL);
pthread_mutex_destroy (&mtx);

pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_lock (&mtx)
pthread_mutex_unlock (&mtx)
```

→ dacă o resursă critică se modifică, trebuie să rezolvi și scrierile pentru că altfel poate citi pe jumătate

```
pthread_rwlock • rwl = ..
pthread_rdlock (&rwl) → cei care verifică
pthread_rwlock (&rwl) → cel care scrie
```

↳ ~~dacă vrem și de citit și scris~~
las oricâți cititori / un singur
0 scriitor / scriitor 0 cititori

1

Merge pt. că int și void* au aceeași dimensiune

```
#include <stdio.h>
# Include <pthread.h>
# define  N 10

void* f (void* a ){          (int) a
      printf ("%d\n", *(int*) a );
      return NULL;
}


int main (int argc, char** argv){
    pthread_t t[N];
    int i;

    for (i=0; i<N; i++)
          pthread_create (&t[i], NULL, f, &i );
    }                                  (void*)i
    for (i=0; i<N; i++) {
          pthread_join ( t[i] , NULL );
    }
    return 0;
}
```

trimite adresa lui i și nu va tipări ce trebuie

Andreo Alexandrescu — Advanced C++
↳ ?? pointers-compilers ??

```
while (1) {

    r = get req();
    if (fork()==0){
         serv(r);
         exit(0);
    }   }
```

```
while(1){
    r = get_req();
    pthread_create (&t, NULL, serve, r);
}
```

MO

OwO