SISTEME DE OPERARE
– Seminar 5 –

PROGRAMARE CONCURENTĂ ÎN UNIX

1. NOȚIUNI GENERALE

- proces = *un calcul care poate fi executat concurent sau în paralel cu alte calcule*
  = *un program aflat în execuție*
- thread (fir de execuție) = *entitate de execuție din cadrul unui proces, compusă dintr-un context și o secvență de instrucțiuni de executat*
- un thread execută o procedură sau o funcție, în cadrul aceluiași proces, concurent sau în paralel cu alte thread-uri
- toate firele de execuție din cadrul unui proces partajează același spațiu de adrese
- toate firele de execuție ale unui proces utilizează în comun instrucțiunile, majoritatea datelor și contextul de execuție
- fiecare fir de execuție are un identificator unic (TID), un set de regiștri și o stivă proprie
- singurul spațiu ocupat exclusiv de către un fir de execuție este spațiul de stivă
- avantajele utilizării firelor de execuție:
  - crearea unui fir de execuție durează mai puțin decât crearea unui proces
  - distrugerea unui fir de execuție durează mai puțin decât distrugerea unui proces
  - trecerea de la un fir de execuție la altul (context switch) este foarte rapidă
  - comunicarea între firele de execuție produce o încărcare (overhead) mai mică a sistemului

2. THREAD-URI POSIX

- implementate de către biblioteca `pthreads (POSIX threads)`
- crearea unui thread (vezi `man 3 pthread_create`):

```
#include <pthreads.h>

int pthread_create(pthread_t *tid, pthread_attr_t *attr,
                   void*(*start_routine)(void *), void *arg);
```

- așteptarea terminării unui thread (vezi `man 3 pthread_join`):

```
#include <pthreads.h>

int pthread_join(pthread_t tid, void **retval);
```

- terminarea unui thread (vezi `man 3 pthread_exit`):

```
#include <pthreads.h>

void pthread_exit(void *retval);
```

- transmiterea unei cereri de abandon unui thread (vezi `man 3 pthread_cancel`):

```
#include <pthreads.h>

int pthread_cancel(pthread_t tid);
```

- exemple: `/exemple/threads/thread_1.c, thread_2.c, thread_3.c, thread_4.c`
- compilare:

```
gcc -Wall -o thread_1 thread_1.c -lpthread
gcc -pthread -Wall -o thread_1 thread_1.c
```

3. SINCRONIZAREA FIRELOR DE EXECUȚIE ÎN LINUX

OBIECTE UTILIZATE PENTRU SINCRONIZARE

Obiect	Declarare	Creare/distrugere	Operații
Semafor	`sem_t`	`sem_init()` `sem_destroy()`	`sem_wait()` `sem_post()`
Mutex (MUtual EXclusion)	`pthread_mutex_t`	`pthread_mutex_init()` `pthread_mutex_destroy()`	`pthread_mutex_lock()` `pthread_mutex_unlock()`

RW lock
(Read-Write lock)
	`pthread_rwlock_t`	`pthread_rwlock_init()`
		`pthread_rwlock_destroy()`
			`pthread_rwlock_wrlock()`
			`pthread_rwlock_rdlock()`
			`pthread_rwlock_unlock()`

Barieră (Barrier)	`pthread_barrier_t`	`pthread_barrier_init()` `pthread_barrier_destroy()`	`pthread_barrier_wait()`

Variabilă de condiție
(Condition variable)
	`pthread_cond_t`	`pthread_cond_init()`
		`pthread_cond_destroy()`
			`pthread_cond_wait()`
			`pthread_cond_signal()`
			`pthread_cond_broadcast()`

3.1. SEMAFOARE POSIX

- implementate de către sistemul de operare
- crearea unui semafor (vezi `man 3 sem_init`):

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- blocarea/deblocarea unei resurse folosind un semafor (vezi `man 3 sem_wait`):

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

- distrugerea unui semafor (vezi `man 3 sem_destroy`):

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

- exemplu: `/exemple/locks/lock_3.c`

3.2. MUTEX

- implementat de către biblioteca `pthreads`

- crearea unui mutex (vezi `man 3 pthread_mutex_init`):

```
#include <pthreads.h>

int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
```

sau prin alocare statică:

```
#include <pthreads.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

- blocarea/deblocarea unei resurse folosind un mutex (vezi `man 3 pthread_mutex_lock`):

```
#include <pthreads.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- distrugerea unui mutex (vezi `man 3 pthread_mutex_destroy`):

```
#include <pthreads.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- exemplu: `/exemple/locks/lock_2.c`

3.3. RW (READ-WRITE) LOCK

- implementat de către biblioteca `pthreads`
- crearea unui RW lock (vezi `man 3 pthread_rwlock_init`):

```
#include <pthreads.h>

int pthread_rwlock_init(pthread_rwlock_t *rwlock,
                        const pthread_rwlockattr_t *attr);
```

- blocarea pentru citire/scriere sau deblocarea unei resurse folosind RW lock (vezi `man 3 pthread_rwlock_rdlock`):

```
#include <pthreads.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

- distrugerea unui RW lock (vezi `man 3 pthread_rwlock_destroy`):

```
#include <pthreads.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
```

- exemplu: `/exemple/locks/lock_4.c`

3.4. BARIERA

- implementată de către biblioteca `pthreads`
- crearea unei bariere (vezi `man 3 pthread_barrier_init`):

```
#include <pthreads.h>

int phtread_barrier_init(pthread_cond_t *restrict barr,
                         const pthread_barrierattr_t *attr, unsigned count);
```

- utilizarea unei bariere (vezi `man 3 pthread_barrier_wait`):

```
#include <pthreads.h>

int pthread_barrier_wait(pthread_barrier_t *barr);
```

- distrugerea unei bariere (vezi `man 3 pthread_barrier_destroy`):

```
#include <pthreads.h>

int pthread_barrier_destroy(pthread_barrier_t *barr);
```

- exemplu: `/exemple/barrier.c`

3.5. VARIABILA DE CONDIȚIE

- implementată de către biblioteca `pthreads`
- crearea unei variabile de condiţie (vezi `man 3 pthread_cond_init`):

```
#include <pthreads.h>

int phtread_cond_init(pthread_cond_t *cond,
                      const pthread_condattr_t *attr);
```

sau prin alocare statică:

```
#include <pthreads.h>

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

- blocarea/deblocarea folosind o variabilă de condiţie (vezi `man 3 pthread_cond_wait`):

```
#include <pthreads.h>

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- distrugerea unei variabile de condiţie (vezi `man 3 pthread_cond_destroy`):

```
#include <pthreads.h>

int pthread_cond_destroy(pthread_cond_t *cond);
```

- exemplu: `/exemple/cond_var.c`

REFERINȚE:

- Curs: `http://www.cs.ubbcluj.ro/~rares/course/os/`
- POSIX threads:

  `http://pages.cs.wisc.edu/~travitch/pthreads_primer.html`

  `https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html`
- Thread synchronization:

  `http://www.informit.com/articles/article.aspx?p=2085690&seqNum=6`

https://docs.oracle.com/cd/E19120-01/open.solaris/816-5137/index.html