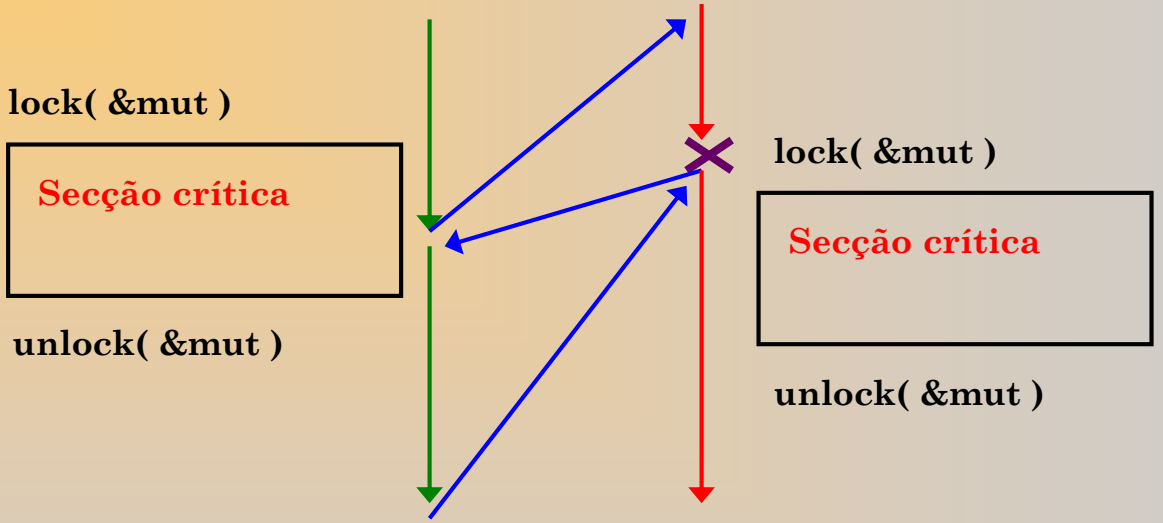
Programação de sistema UNIX

Sincronização:

- mutexes
- semáforos
- variáveis de condição

Mutexes

- Os mutexes em UNIX pertencem à API de threads
- Servem principalmente para proteger secções críticas de acesso a variáveis partilhadas
- Existem apenas em 2 estados: **locked** e **unlocked** (são equivalentes a semáforos inicializados em 1)




```
#include <pthread.h>
```

ficheiro de inclusão

```
pthread_mutex_t  mutex  = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_init( pthread_mutex_t *mut,  pthread_mutexattr_t *attr);
int pthread_mutex_lock( pthread_mutex_t *mut);
int pthread_mutex_trylock( pthread_mutex_t *mut);
int pthread_mutex_unlock( pthread_mutex_t *mut);
int pthread_mutex_destroy( pthread_mutex_t *mut);
```

dispensa

NULL

operações

0
EBUSY

Exemplo

```
int x = 0, inc = 1;

void *modify(void *a) {
  while (1) {
    if (x = = 5)
      inc = -1;
    if (x = = -5)
      inc = 1;
    x = x + inc;
  }
}

int main(void) {
  int k;
  pthread_t tid;

  for (k=0; k<5; k++)
    pthread_create(&tid, NULL,
                   modify, NULL);
  while (1) {
    printf("%d\n", x);
  }
  return 0;
}
```

```
int x = 0, inc = 1;
pthread_mutex_t mut =
                   PTHREAD_MUTEX_INITIALIZER;

void *modify(void *a) {
  while (1) {
    pthred_mutex_lock (& mut);
    if (x = = 5)
      inc = -1;
    if (x = = -5)
      inc = 1;
    x = x + inc;
    pthread_mutex_t unlock( & mut);
  }
}

int main(void) {
  int k;
  pthread_t tid;

  for (k=0; k<5; k++)
    pthread_create(&tid, NULL, modify, NULL);
  while (1) {
    printf("%d\n", x);
  }
  return 0;
}
```

Semáforos POSIX

```
#include <semaphore.h>
```

se 1
variável sem em memória partilhada

```
sem_t sem;
```

usualmente 0

sem nome
(usualmente 1 só processo)

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

ou

O_CREAT
O_EXCL
0

com nome
(vários processos)

```
sem_t * sem_open ( char * name, int flags, ...);
int sem_close(sem_t *sem);
int sem_unlink ( char * name );
```

mode_t mode, unsigned int value

inicialização
e
libertação

```
int sem_wait ( sem_t * sem );
int sem_trywait ( sem_t * sem );
int sem_getvalue ( sem_t * sem,  int * value );
int sem_post ( sem_t * sem );
```

operações

Variáveis de condição

Utilizadas para bloquear um *thread* até se verificar uma condição e entrar protegido numa Secção Crítica

```
while(1) {
   pthread_mutex_lock(&mut);
   if (x == y)
      break;
   pthread_mutex_unlock(&mut);
}
.....        /* secção crítica */
pthread_mutex_unlock(&mut);
```

Espera pela condição (x == y) em busy-waiting

utilizando uma variável de condição

```
pthread_mutex_lock(&mut);
while (x != y)
   pthread_cond_wait(&var, &mut);
.....        /* secção crítica */
pthread_mutex_unlock(&mut);
```

Outro *thread*

```
modifica x e/ou y numa secção crítica
```

Outro *thread* que modifica x ou y

```
pthread_mutex_lock(&mut);
... ...  /* modifica o valor de x ou y ou ambos */
pthread_cond_signal(&var);
pthread_mutex_unlock(&mut);
```

Liberta um *thread* bloqueado

A API para variáveis de condição

```
#include <pthread.h>
```

ficheiro de inclusão

```
pthread_cond_t  cvar = PTHREAD_COND_INITIALIZER;
```

dispensa

NULL

operações

```
int pthread_cond_init ( pthread_cond_t  * cvar,  pthread_condattr_t  * attr );
int pthread_cond_wait ( pthread_cond_t  * cvar,  pthread_mutex_t  * mut );
int pthread_cond_signal ( pthread_cond_t  * cvar );
int pthread_cond_broadcast ( pthread_cond_t  * cvar );
int pthread_cond_destroy ( pthread_cond_t  * cvar );
```

Os mutexes e as variáveis de condição **poderão ser partilhados entre processos** se forem criados em memória partilhada e inicializados com um atributo contendo a propriedade:

PTHREAD_PROCESS_SHARED

```
pthread_condattr_t attrs;

pthread_condattr_init(&attrs);
pthread_condattr_setpshared(&attrs, PTHREAD_PROCESS_SHARED);
```

Exemplo

```
int x = 0, y = 10;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cvar = PTHREAD_COND_INITIALIZER;

void *incr(void *a)
{
   while (1) {
      pthread_mutex_lock(&mut);
      x = x + 1;
      pthread_cond_signal(&cvar);
      pthread_mutex_unlock(&mut);
   }
}

void *test(void *a)
{
   while (1) {
      pthread_mutex_lock(&mut);
      while  (x != y)
          pthread_cond_wait(&cvar, &mut);
      printf("x = y = %d\n", x);
      x = 0;
      y = y + 10;
      pthread_mutex_unlock(&mut);
   }
}
```

modifica uma das variáveis que intervêm na condição

Várias instâncias deste *thread*

Uma instância