Нишки (threads)

доц. д-р инж. Христо Вълчанов

http://cs.tu-varna.bg

Нишки

Нишка – независим поток от инструкции в рамките на една програма:

- Има различни състояния (running, ready и др.).
- Контекстът на нишката се съхранява когато тя не се изпълнява.
- Нишките се планират от ОС и се изпълняват като независими единици в рамките на процес.
- Нишките имат собствен стек;
- Един процес може да има множество нишки;
- Нишка има достъп до адресното пространство и ресурсите на процеса.

Процеси / нишки

- Множество нишки имат достъп до едно и също адресно пространство.
- Всяка нишка се представя чрез структура - thread control block (TCB).
- Всяка нишка може да създава други нишки.

Предимства на нишките

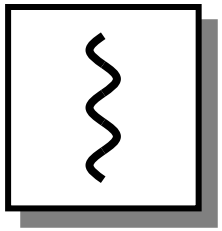
- По-малко време за създаване.
- По-малко време за завършване.
- По-малко време за превключване между нишките.
- По-малко комуникационни разходи.

Multi-threading / Single threading

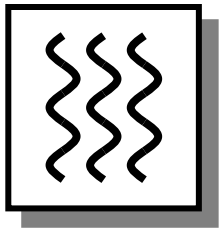
- **Single threading**

  ОС не разпознава нишки.
- **Multi-threading**

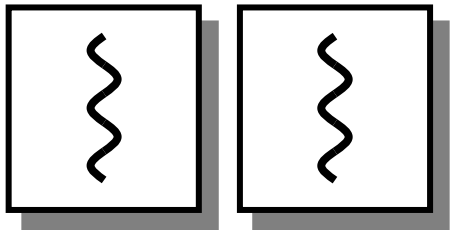
  Поддържа нишки.



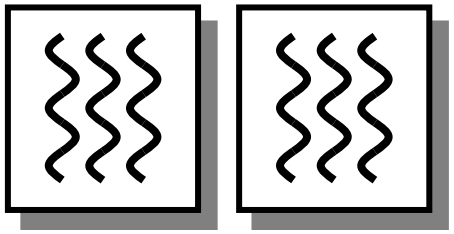
single process, single thread



single process, multiple threads



multiple processes, single thread

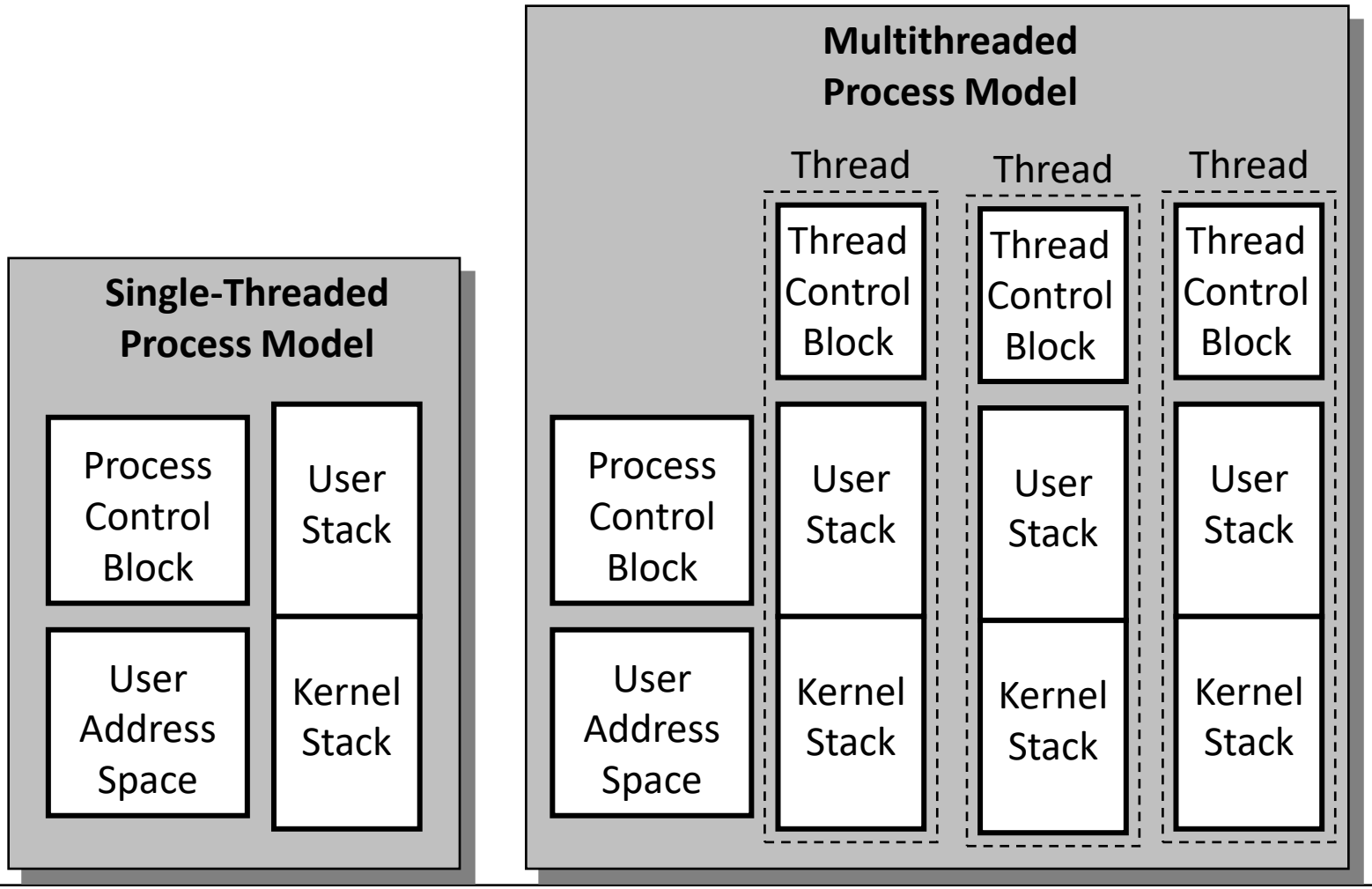


multiple processes, multiple threads

Примери на ОС:

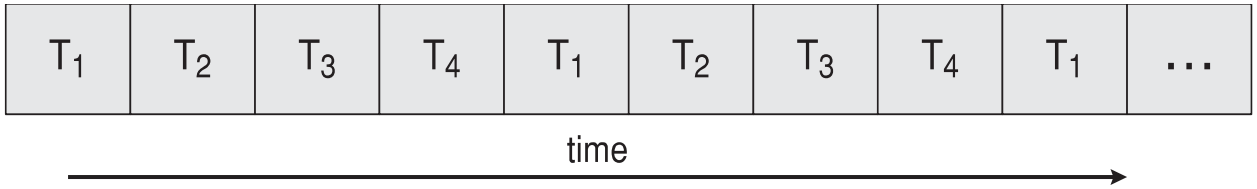
- **MSDOS** – един процес и една нишка
- **UNIX** – множество процеси с по една нишка всеки
- **Solaris, Windows** – множество нишки

Модели на single и multithread приложения

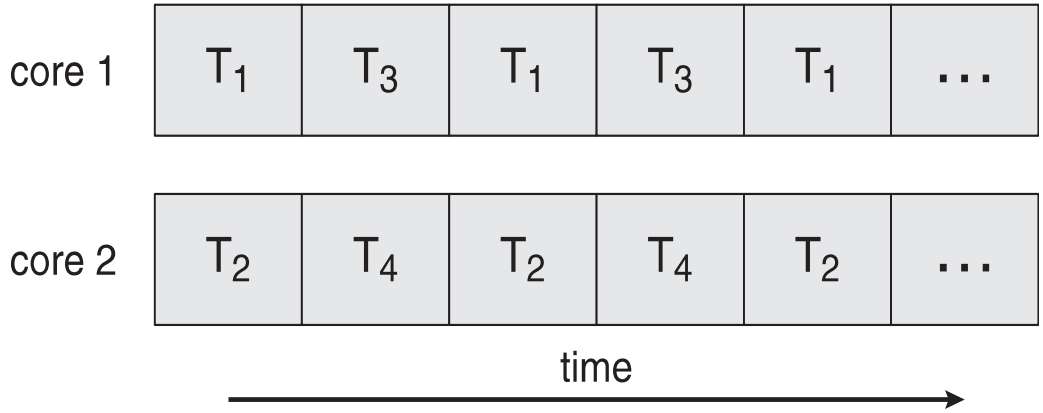


Паралелно и конкурентно изпълнение

- Конкурентно изпълнение в едноядрена система



- Паралелно изпълнение в многоядрена система



Multi-threaded клиент: Web Browsers

Предимства на многонишковото програмиране

- Подобрява производителността на приложенията.
- Използване на многоядрени процесори.
- Подобрява структурата на програмите.
- Използват се по-малко системни ресурси.

Реализация на нишки

- **User level** – ядрото не поддържа нишки. Управлението им е чрез библиотека.
- K**ernel level** – управлението на нишките се осигурява от ядрото.

Библиотеки за нишки

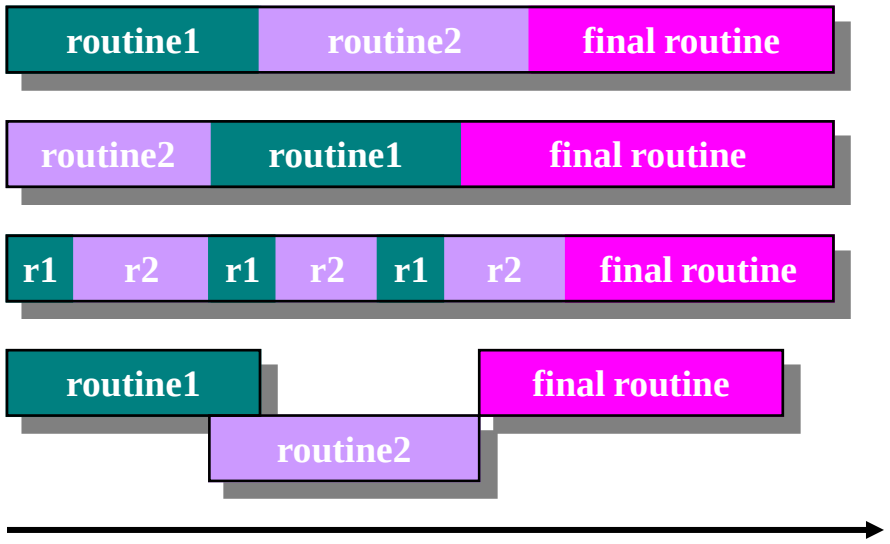
- Предоставят интерфейс за работа с нишки.
  - **libpthread** for POSIX threads.
  - **libthread** for Solaris threads.
  - Java Threads
  - Win 32 Threads

Пакети за нишки

- Posix Threads (pthreads)
  - Широко използван
  - Базиран на Posix standard
  - Опростени извиквания: *pthread_create*, …
  - Типично използван в C/C++ приложения
  - Може да бъде реализиран и като user-level и като kernel-level
- Java Threads
  - Естествена поддръжка, вградена в езика
  - Нишките се планират от JVM
- Win 32 Threads

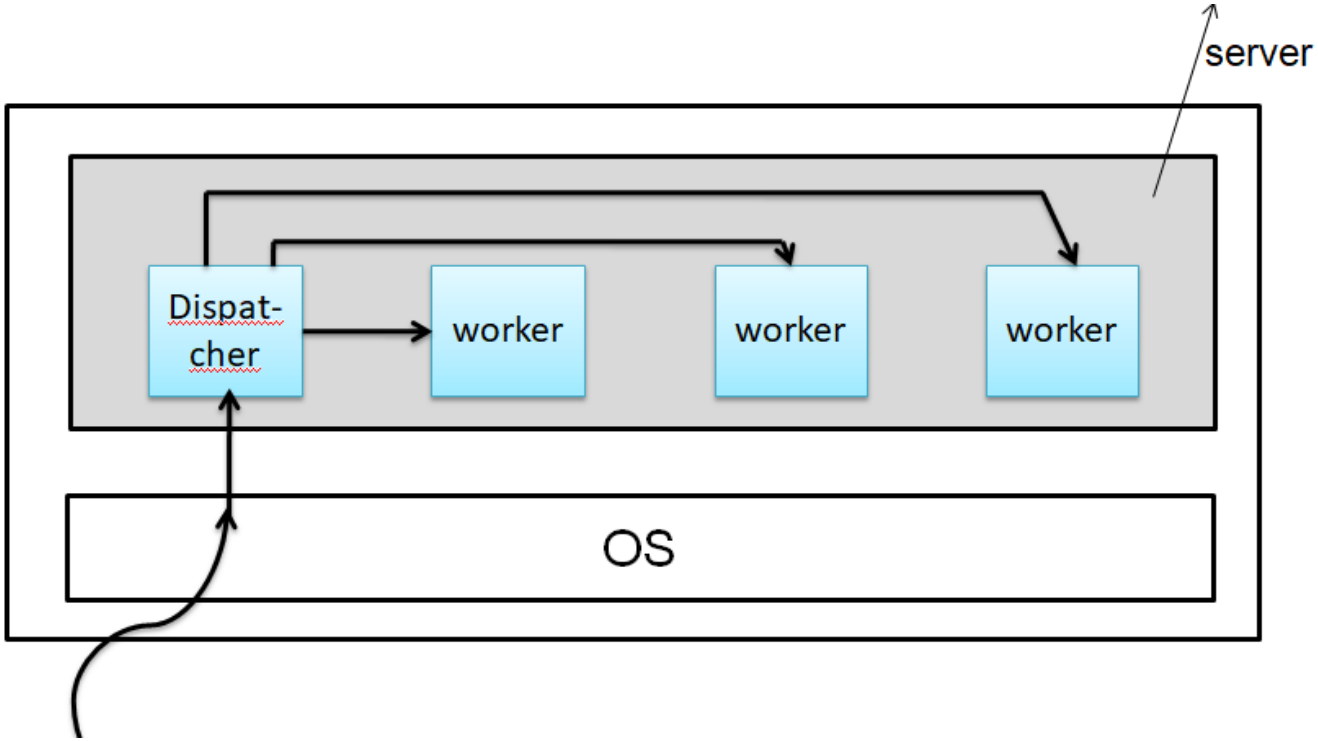
Проектиране на многонишкови програми

- Ако програмата може да се организира в независими, конкурентно изпълняващите се единици.



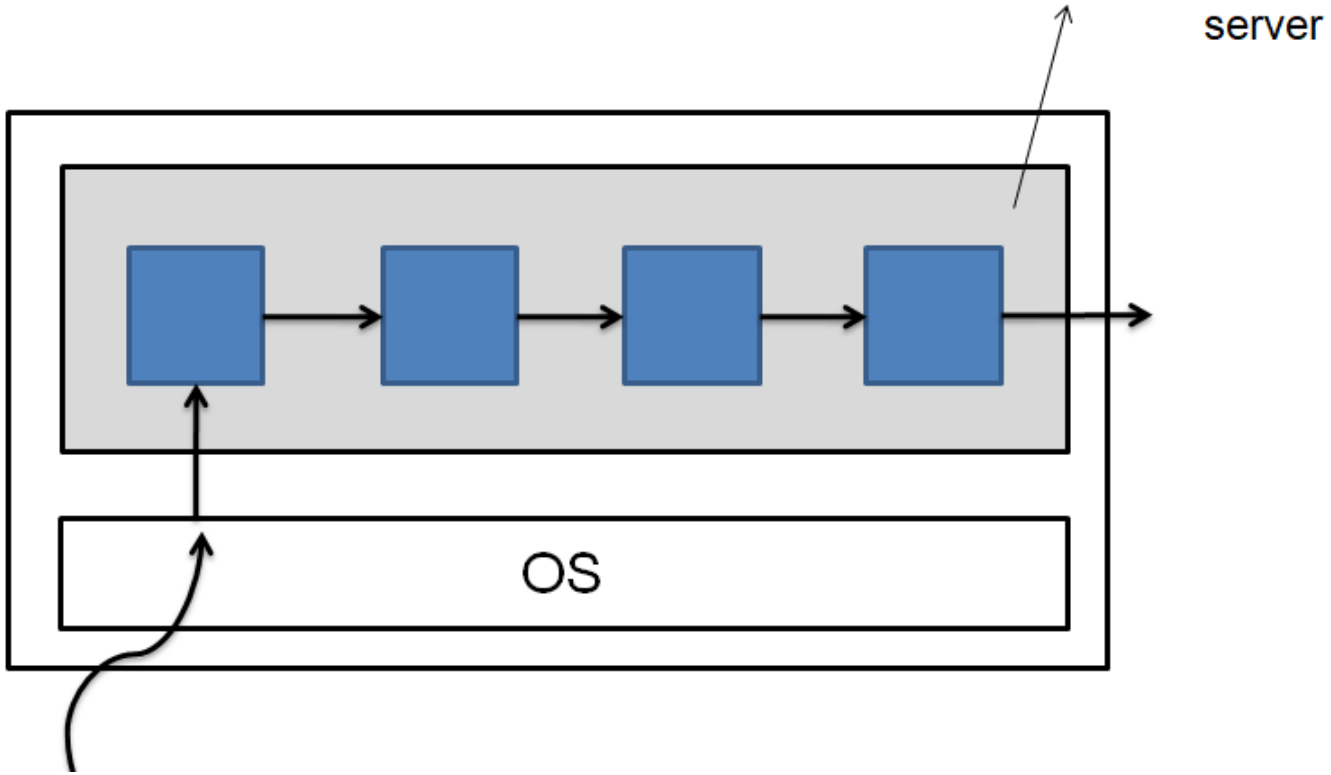
Общи модели на многонишкови програми

- **Manager-worker**



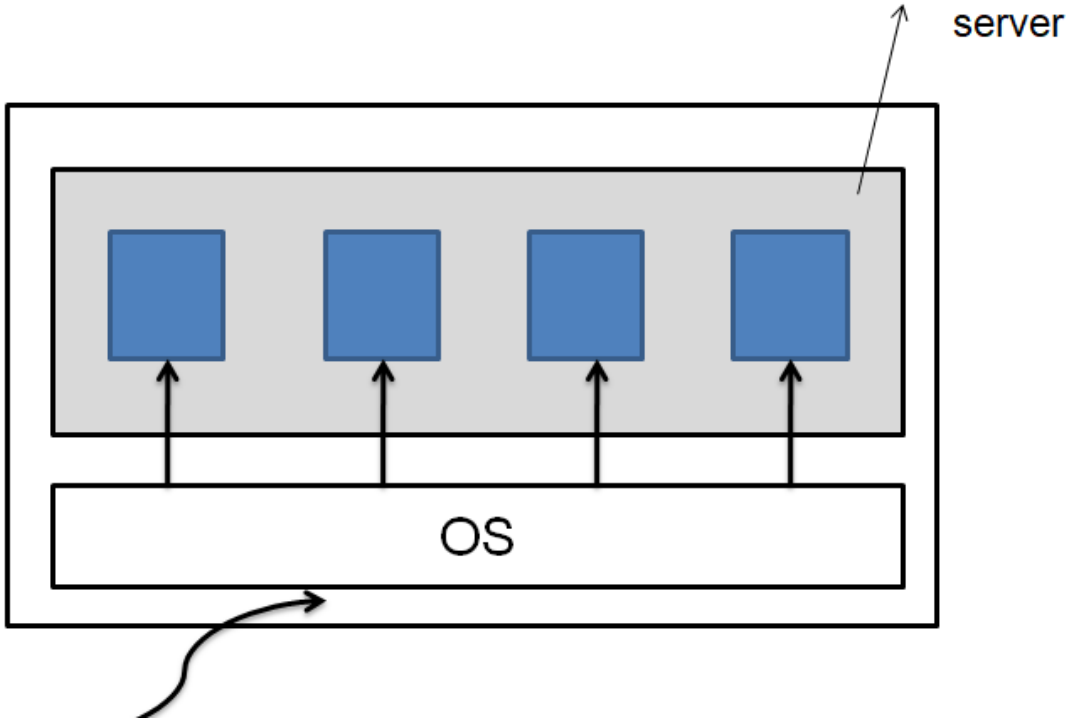
Общи модели на многонишкови програми

- **Pipeline**



Общи модели на многонишкови програми

- **Team server**



Pthreads API

- **Управление на нишки**
- **Mutexes**
- **Conditional variables**

Създаване на нишка

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

Завършване на нишка

- При завършване на функцията, от която е създадена;
- При извикване на *pthread_exit();*
- Процесът завършва чрез *exit().*

```
void pthread_exit(void *status);
```

Пример

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid){
      printf("\n%d: Hello World!\n", threadid);
      pthread_exit(NULL);
}

int main (int argc, char* argv[]){
      pthread_t threads[NUM_THREADS];
      int rc, t;

      for(t=0; t < NUM_THREADS; t++){
         printf("Creating thread %d\n", t);

        // Create a thread and pass its number as argument
          rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);

         if (rc){
           printf("ERROR: pthread_create %d\n", rc);
           exit(1);
         }
      }

      pthread_exit(NULL);
}
```

Пример 2

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

 void *Plus(void *threadid){
      int k;
          for ( ; ; ) {
            printf("+");
            for (k=0;k<10000;k++);
          }
}
 void *Minus(void *threadid){
      int k;
          for ( ; ; ) {
            printf("-");
            for (k=0;k<10000;k++);
          }
}
 void *Dot(void *threadid){
      int k;
          for ( ; ; ) {
            printf(".");
            for (k=0;k<10000;k++);
          }
}
```

```
int main (int argc, char* argv[]){
     pthread_t t1, t2, t3;

     printf("Creating thread %d\n", t);

     pthread_create(&t1, NULL, Plus, NULL);
     pthread_create(&t2, NULL, Minus, NULL);
     pthread_create(&t3, NULL, Dot, NULL);

     pthread_exit(NULL);
}
```

Изчакване на нишка

```
int pthread_join(thread_t tid,
                 void **status);
```

Пример

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     3

// A working thread
void *BusyWork(void *null) {
   int i;
   double result=0.0;

   // Simulate working
   for (i=0; i < 1000000; i++) {
     result = result + (double)random();
   }
   printf("result = %e\n",result);
   pthread_exit(NULL);
}
```

Пример

```
int main (int argc, char* argv[]) {
      pthread_t thread[NUM_THREADS];
      int rc, t, status;

   for(t=0; t < NUM_THREADS; t++) {
         printf("Creating thread %d\n", t);

         // Create a thread as JOINABLE
         rc = pthread_create(&thread[t],
                             NULL, BusyWork, NULL);
         if (rc) {
            printf("ERROR: pthread_create %d\n",rc);
            exit(1);
         }
   }
```

Пример

```
   for(t=0;t < NUM_THREADS;t++) {
          // Wait for other treads
          rc = pthread_join(thread[t], (void *)&status);
          if (rc) {
             printf("ERROR: pthread_join %d\n", rc);
             exit(1);
          }
          printf("Main: Completed join with thread %d
                              status= %d\n",t,status);
   }

   pthread_exit(NULL);
}
```

Синхронизация на нишка

- Mutual Exclusion locks (Mutex).
- Condition variables.
- Semaphores.

Mutexes - операции

- **lock()**
- **unlock()**
- **trylock()**

Използване

```
lock mutex;
    Critical Section;
unlock mutex;
```

Инициализиране и унищожаване на mutex

```
int pthread_mutex_init(pthread_mutex_t *mp,
                       const pthread_mutexattr_t *mattr);
```

- ✓ **mutex е първоначално отключен**

```
int pthread_mutex_destroy(pthread_mutex_t *mp);
```

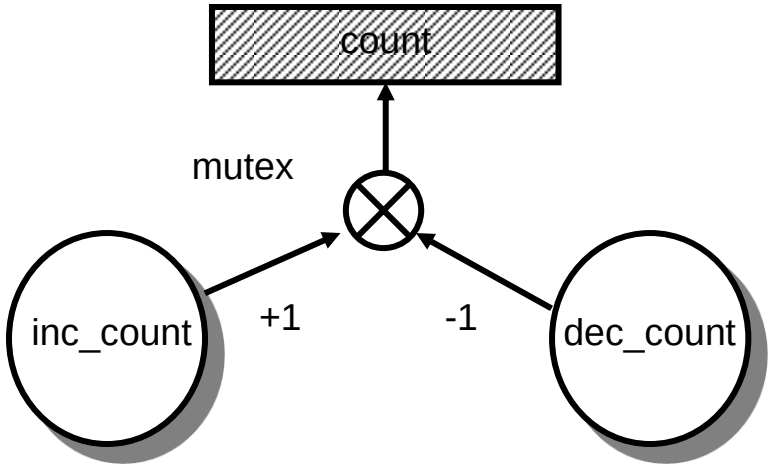
Заключване/отключване на mutex

```
int pthread_mutex_lock(pthread_mutex_t *mp);

int pthread_mutex_unlock(pthread_mutex_t *mp);
```

Пример

```
#include <stdio.h>
#include <pthread.h>
int count=0;
pthread_mutex_t mu;      // Mutex variable
void *inc_count(void* arg) {
     int i=0;
     for (i=0;i<1000000;i++) {
         pthread_mutex_lock(&mu);
         count++;
         pthread_mutex_unlock(&mu);
         if (i%100000==0) printf("+\n");
     }
     pthread_exit(NULL);
}
void *dec_count(void* arg) {
     int i=0;
     for (i=0;i<1000000;i++) {
         pthread_mutex_lock(&mu);
         count--;
         pthread_mutex_unlock(&mu);
         if (i%100000==0) printf("-\n");
     }
     pthread_exit(NULL);
}
```



```
int main(int argc, char* argv[]) {
     pthread_t t1, t2;

     // Initialize the mutex
     pthread_mutex_init(&mu, NULL);

     pthread_create(&t1, NULL, inc_count, NULL);
     pthread_create(&t2, NULL, dec_count, NULL);

     pthread_join(t1, NULL);
     pthread_join(t2, NULL);

     // Free the mutex resources
     pthread_mutex_destroy(&mu);

     printf("Main: Done.\n");

     pthread_exit(NULL);
}
```

Семафори - операции

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

int sem_wait(sem_t *sem);

int sem_post(sem_t *sem);

int sem_getvalue(sem_t *sem, int *sval);
```

Пример

```
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#define NUMTHREADS        5
sem_t semaphore; // The semaphore object
// Thread routine to wait on a semaphore.
 void *sem_waiter (void *arg){
   int *id = (long)arg;
    printf("Thread %d waiting\n", *id);
    sem_wait(&semaphore);
    printf("Thread %d resuming\n", *id);
    pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[]){
     int thread_count;
     int sem_value;
     pthread_t sem_waiters[NUMTHREADS];
     int status;
    // Initialize the semaphores
     if (sem_init(&semaphore, 0, 0) == -1) {
       perror("Init semaphore");
        exit(1);
    }
    // Create 5 threads to wait concurrently on the semaphore.
     for (thread_count=0;thread_count<NUMTHREADS; thread_count++)
     {
      status = pthread_create(&sem_waiters[thread_count], NULL,
                              sem_waiter, (void*)thread_count);
      if (status != 0) {
         perror("Create thread");
         exit(1);
      }
    }
```

```
// "Broadcast" the semaphore by repeatedly posting until
// the count of waiters goes to 0.
   while (1) {
     if (sem_getvalue(&semaphore, &sem_value) == -1) {
         perror("Get semaphore value");
         exit(1);
     }
     if (sem_value > 0) break;
     printf ("Posting from main: %d\n", sem_value);
     if (sem_post(&semaphore) == -1) {
     perror("Post semaphore");
     exit(1);
     }
   }
```

Въпроси?