POSIX нишки

доц. д-р инж. Христо Вълчанов

http://cs.tu-varna.bg

Pthreads API

- **Управление на нишки**
- **Mutexes**

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);
}
```

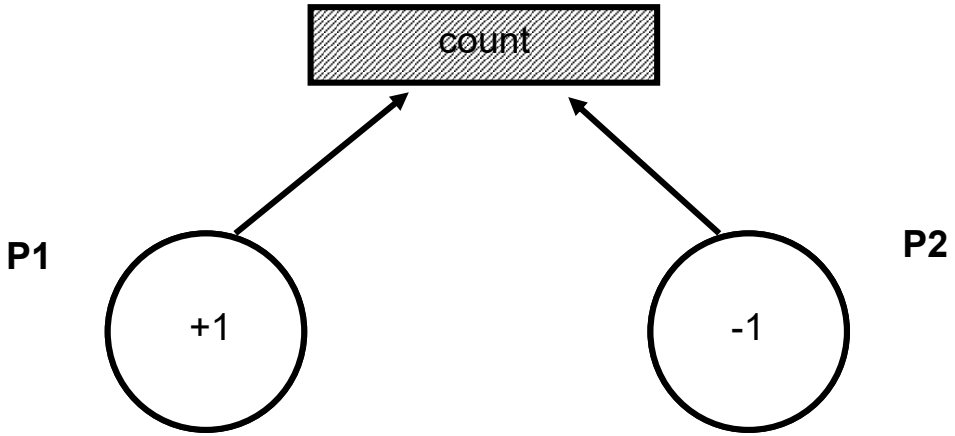
Критична секция

- **Критична секция (Critical Section - CS)** – тази част от кода на нишка, в който има обръщение към общ ресурс.

- Изисквания към CS:
  - В критична секция в даден момент може да се намира само една нишка.
  - Ако една или няколко нишки желаят да влязат в критичните си секции, то на една от тях това задължително трябва да бъде разрешено.
  - Всяка нишка може да преминава в произволно състояние извън критичната си секция, т.е. може да бъде дори спряна. Това не трябва да оказва влияние върху другите нишки при използване на общия ресурс.
  - Относителните скорости на развитие на нишките са предварително неизвестни и произволни.
  - Нишка не може да спре изпълнението си в критична секция.
  - Критичните секции не трябва да имат приоритети.

Пример за достъп до общ ресурс

Нишка P1 извършва увеличава променливата *count* 1000000 пъти, нишка P2 намалява променливата *count* 1000000 пъти. Началната стойност на *count* е 0.



- **Каква е стойността на `count` след завършване на нишките?**

Пример за достъп до общ ресурс

```
int count;
```

```
void *P1() {

   for (i=0; i<1000000; i++) {
        . . .
        inc(count);
        . . .
   }
}
```

```
void *P2() {

   for (i=0; i<1000000; i++) {
        . . .
        dec(count);
        . . .
   }
}
```

Пример за достъп до общ ресурс

- След компилация:

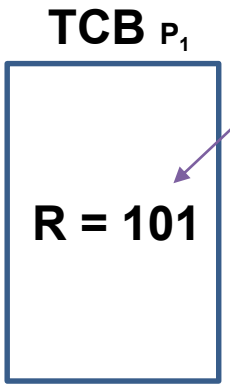
```
P1:     LOADR   count        Зарежда count в регистър R
        INCR                 Инкрементира R
        STORER  count        Записва R на адрес count

P2:     LOADR   count        Зарежда count в регистър R
        DECR                 Декрементира R
        STORER  count        Записва R на адрес count
```

Влияние на прекъсванията

quant	P1	count	R	P2
t_1	. . .	100	???	. . .
t_2	LOADR count INCR	100 100	**100** **101**	

Влияние на прекъсванията (2)

quant	P1	count	R	P2
t_1	. . .	100	???	. . .
t_2	LOADR count INCR	100 100	100 101	



Влияние на прекъсванията (3)

quant	P1	count	R	P2
t_1	. . .	100	???	. . .
t_2	LOADR count INCR	100 100	100 101	
t_3		100 100 **99**	**100** **99** 99	LOADR count DECR STORER count

TCB P_1

R = 101

Влияние на прекъсванията (4)

quant	P1	count	R	P2
t_1	. . .	100	???	. . .
t_2	LOADR count INCR	100 100	100 101	
t_3		100 100 **99**	**100** **99** 99	LOADR count DECR STORER count

прекъсване

TCB P_1

R = 101

TCB P2

R = 99

Влияние на прекъсванията (5)

quant	P1	count	R	P2
t_1	. . .	100	???	. . .
t_2	LOADR count INCR	100 100	100 101	
t_3		100 100 99	100 99 99	LOADR count DECR STORER count
t_4	STORER count	99 **101**	**101** 101	

TCB P_1

R = 101

TCB P_2

R = 99

Синхронизация на нишки

- Mutual Exclusion locks (Mutex).
- Semaphores.

Mutexes - операции

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

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**

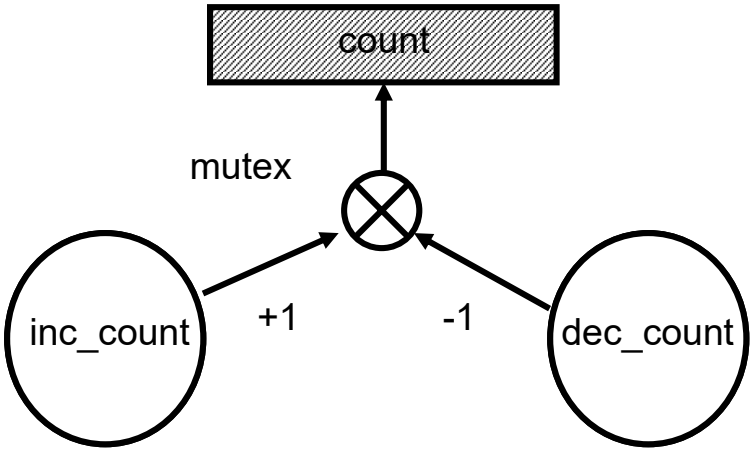
```
int pthread_mutex_destroy(pthread_mutex_t *mp);
```

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: count is %d.\n", count);

    pthread_exit(NULL);
}
```

Семафори

- **Семафор** – специален вид целочислена положителна променлива, върху която са определени две операции (примитиви) P и V.
- Операциите са неделими.
- Семафор, приемащ само две стойности 0 и 1 – **двоичен семафор.**

Семафори

- **операция P(S):**

Нишката се опитва да изпълни S=S-1. При това са възможни два случая:

  - S>0. Нишката изпълнява S=S-1 и продължава своето развитие;
  - S=0. Нишката се блокира (wait) докато S стане по-голямо от нула. Тогава и само тогава тя завършва операцията S=S-1 и продължава изпълнението си.

- **операция V(S):**

Нишката изпълнява операцията S=S+1. При това също са възможни два случая:

  - S>0. Нишката, изпълняваща операцията V(S), продължава да работи;
  - S=0. Нишката, изпълняваща операцията V(S), продължава да работи, но преди това се активира една от блокираните нишки (т.е. дава й се възможност да завърши започнатата си P(S) операция).

Използване на семафори

```
semaphore sem=1;


P1:     for (;;) {
          P(sem);
          CS1;
          V(sem);
          Prog1;
        }
P2:     for (;;) {
          P(sem);
          CS2;
          V(sem);
          Prog2;
        }
```

Семафор с повече от две стойности – общ семафор

Семафори - операции

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

int sem_wait(sem_t *sem); // аналог на P(S)

int sem_post(sem_t *sem); // аналог на V(S)
```

Пример

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

int count=0;
sem_t S; // Semaphore variable
void *inc_count(void* arg) {
      int i=0;
      for (i=0;i<1000000;i++) {
          sem_wait(&S);
          count++;
          sem_post(&S);
          if (i%100000==0) printf("+\n");
      }
      pthread_exit(NULL);
}
void *dec_count(void* arg) {
      int i=0;
      for (i=0;i<1000000;i++) {
          sem_wait(&S);
          count--;
          sem_post(&S);
          if (i%100000==0) printf("-\n");
      }
      pthread_exit(NULL);
}
```

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

     // Initialize the semaphore
     if (sem_init(&S, 0, 1) == -1) {
        perror("Init semaphore");
        exit(1);
     }

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

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

     printf("Main: count is %d.\n", count);

     pthread_exit(NULL);
}
```

Многонишков сървър

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <stdlib.h>

#define SVRPORT        3000

// Worker thread function
void *worker(void *arg) {
  int D1, D2, operation, sum;
  int sd_copy;

  // The argument is the socket copy from accept()
  sd_copy = *(int*)arg;

  // Read from the socket
  read(sd_copy, &D1, sizeof(D1));
  read(sd_copy, &D2, sizeof(D2));
  read(sd_copy, &operation,
              sizeof(operation));
```

```
switch (operation) {
    case 0: sum = D1 + D2; break;
    case 1: sum = D1 - D2; break;
    case 2: sum = D1 * D2; break;
    case 3: sum = D1 / D2; break;
    case 4: exit(0);
     default: {
      printf("Server: Error\n");
      exit(1);
    }
}

  write(sd_copy, &sum, sizeof(sum));
// Return the result

  close(sd_copy);

  pthread_exit(0);//End of the worker
}
```

Многонишков сървър

```
int main() {
  int sd, sd_copy;
  struct sockaddr_in myaddr, theclient;
  int clientlen, tmp=1;
  pthread_t td;

  // Create a socket
  sd = socket(AF_INET, SOCK_STREAM, 0);
  if (sd == -1) {
    perror("socket");
    exit(1);
  }
  // Set socket to be with REUSABLE port
  if ( setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &tmp,
        sizeof(int)) == -1 ) {
    perror("setsckopt");
    exit(1);
  }

  // Bind to the socket
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons(SVRPORT);
  myaddr.sin_addr.s_addr = INADDR_ANY;

  if ( bind(sd, (struct sockaddr *)&myaddr,
        sizeof(myaddr)) == -1 ) {
    perror("bind");
    exit(1);
  }
```

```
  // Prepare buffer for incomming connections
  if ( listen(sd, 5) == -1 ) {
    perror("listen");
    exit(1);
  }

  // Server's main loop
  while (1) {
    // Wait for a client connection
    sd_copy = accept(sd,
          (struct
          sockaddr*)&theclient,
          &clientlen);
    if (sd_copy == -1) {
      perror("accept");
      exit(1);
    }

    // Start worker thread
    pthread_create(&td, NULL, worker,
          (void *)&sd_copy);
  }

  exit(0);
}
```

Въпроси?