

Нишки

Любомир Чорбаджиев¹
lchorbadjiev@elsys-bg.org

¹Технологическо училище “Електронни системи”
Технически университет, София

17 февруари 2016 г.



Съдържание

- 1 Въведение
- 2 Нишки
- 3 Нишки в потребителското пространство
- 4 Нишки в пространството на ядрото
- 5 Комбиниран подход
- 6 Нишки в Java
- 7 Pthreads

Процеси

- Когато операционната система разпределя ресурсите, тя ги разпределя между процесите, които се изпълняват върху компютърната система. Собствеността на ресурсите се държи от процесите.
- Планирането на процесора се извършва в термините на задачи или олекотени процеси.
- Операционната система третира тези два аспекта независимо.

Многонишковост

- Съвременните операционни системи поддържат изпълнението на много нишки в рамките на един процес.
- Ранните операционни системи поддържат само една последователност на изпълнение – един процес с една нишка на изпълнение.
- Ранния UNIX поддържа много процеси, но всеки процес поддържа само по една нишка.
- Съвременните операционни системи – Solaris, Linux, Windows XP, Mac OS X и т.н. – имат пълна поддръжка на многонишковост.

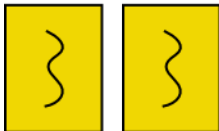
Многонишковост



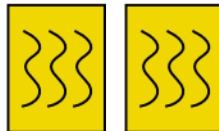
**Един процес,
една нишка**



**Един процес,
много нишки**



**Много процеси,
една нишка**

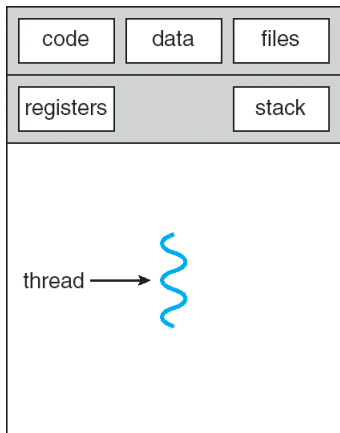


**Много процеси,
много нишки**

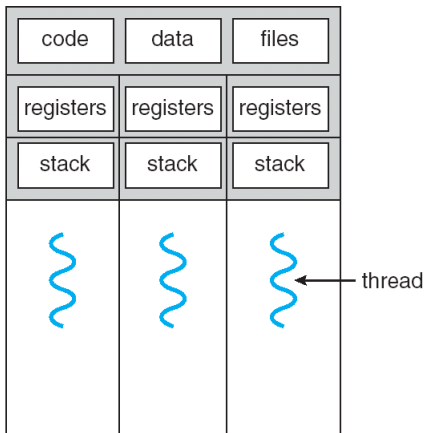
Нишки

- Всяка нишка притежава състояние – в изпълнение, готова, и т.н.
- Когато не се изпълнява, за всяка нишка се съхранява контекст на нишката – контролен блок на нишката.
- Всяка нишка притежава собствен стек за изпълнение.
- Нишките имат достъп до паметта и всички ресурси на процеса в който се изпълняват. Ресурсите на процеса са общи за всички нишки.
- Нишките притежават и локална за нишките памет – могат да съхраняват стойности, които са локални за нишката (не се виждат от другите нишки).

Нишки



single-threaded process



multithreaded process

Фигура : Silberschatz, Gavin, Gagne: *Operating Systems Concepts*

Предимства от използването на нишки

- Използването на нишки води до икономия на ресурси на компютърната система – олекотени процеси.
- Създаването на нишка и унищожаването на нишка изисква по-малко време.
- Превключването на контекста между нишки става по-бързо.
- Нишките, които са част от един и същ процес, си поделят адресното пространство и файловете, и поради това те могат да комуникират без обръщане към ядрото.
- Подобрява се използването на многопроцесорните архитектури.

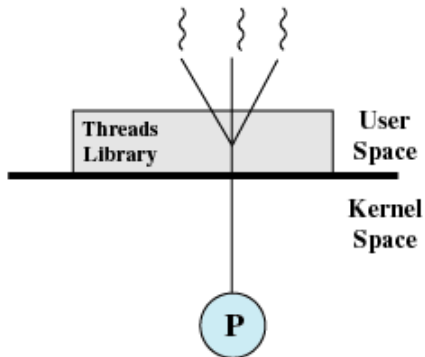
Нишки

- Спирането на процеса води до спиране на всички нишки на процеса, тъй като всички нишки на процеса си поделят адресното пространство.
- Унищожаването на процеса води до унищожаването на всички нишки на процеса.
- Състояния на нишките:
 - Нова – размножаване на нишката.
 - Блокирана.
 - Изпълнява се.
 - Унищожена – освобождава се контекста на нишката и стека.

Нишки в потребителското пространство

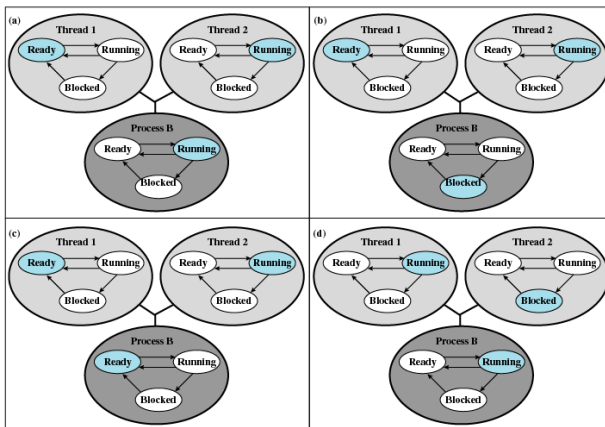
- Нишките в потребителския процес съответстват на единствена нишка в ядрото.
- Управлението на нишките се извършва от библиотека, която изцяло работи в потребителското пространство.
- Ядрото по никакъв начин не знае и не се интересува от съществуването на нишките.
- Има редица примери за библиотеки, които поддържат нишки в потребителското пространство:
 - Solaris Green Threads.
 - GNU Portable Threads.
 - WIN32 Threads.

Нишки в потребителското пространство



Фигура : Stallings: *Operating Systems: Internals and Design Principles*

Нишки в потребителското пространство



Colored state
is current state

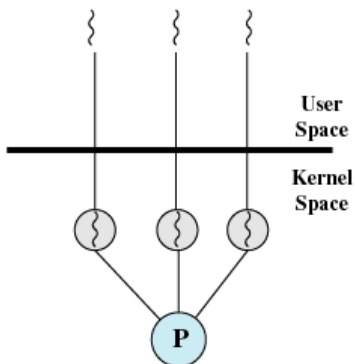
Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Фигура : Stallings: *Operating Systems: Internals and Design Principles*

Нишки в пространството на ядрото

- Всяка нишка в потребителското пространство съответстват на отделна нишка в ядрото.
- Изискват поддръжка в ядрото.
- Ядрото поддържа информация за контекста на процесите и на нишките.
- Планирането на работата на процесора е на ниво нишки.
- Примери:
 - Solaris.
 - Linux.
 - Windows XP/2000.
 - Tru64 UNIX.
 - Mac OS X.

Нишки в пространството на ядрото



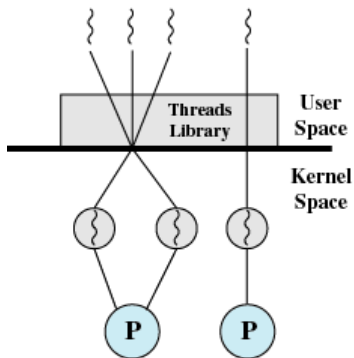
(b) Pure kernel-level

Фигура : Stallings: *Operating Systems: Internals and Design Principles*

Комбиниран подход

- Позволява много потребителски нишки да съответстват на много нишки от пространството на ядрото.
- Операционната система създава достатъчен брой нишки на ядрото.
- Планирането и синхронизацията на нишките става в рамките на потребителското пространство.

Комбиниран подход

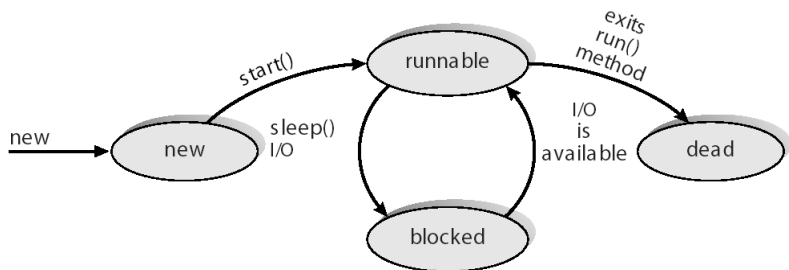


(c) Combined

Фигура : Stallings: *Operating Systems: Internals and Design Principles*

Нишки в Java

- Нишките в Java се управляват от JVM.
- Има два начина да се реализират нишки в Java: чрез наследяване на класа Thread или чрез имплементация на интерфейса Runnable.



Фигура : Silberschatz, Gavin, Gagne: *Operating Systems Concepts*

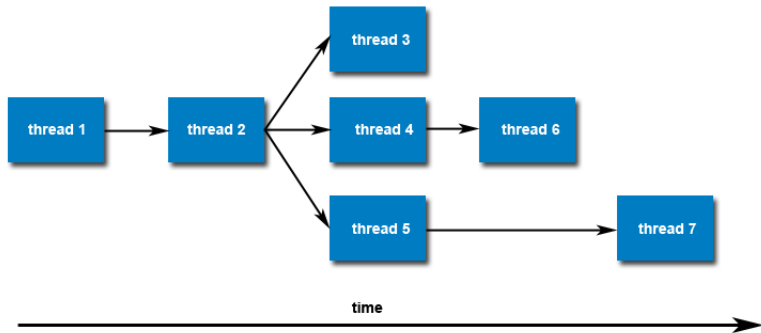
Pthreads

- POSIX – стандарт IEEE 1003.1c.
- Дефинира стандартно API за създаване на нишки и синхронизация.
- POSIX специфицира поведението на библиотеката.
- Реализацията е въпрос разработка.
- POSIX Pthreads е стандартна за UNIX операционните системи – Solaris, Linux, Mac OS X.
- <http://www.llnl.gov/computing/tutorials/pthreads/>

Създаване на нишки

- Първоначално главната функция `main()` разполага с единствена нишка – нишката на процеса. Всички останали нишки трябва изрично да бъдат създадени.
- Функцията `pthread_create()` се използва за създаване на нова нишка. Тази функция може да се вика произволен брой пъти от всяка точка на програмата.
- Максималният брой нишки, които могат да бъдат стартирани от един процес зависи от имплементацията на библиотеката.
- Веднъж създадени, нишките са равнопоставени, и могат да създават други нишки. Няма йерархия или зависимост между нишките.

Създаване на нишки



Фигура : <http://www.llnl.gov/computing/tutorials/pthreads/>

Създаване на нишки

```
1 int pthread_create(pthread_t *thread,  
2     const pthread_attr_t *attr,  
3     void *(*start_routine)(void*),  
4     void *arg);
```

Аргументите на `pthread_create()` са:

- `thread` – уникален идентификатор на създадената нишка.
- `attr` – специфицира атрибутите на нишката.
- `start_routine` – функция, която се изпълнява от нишката при нейното стартиране.
- `arg` – аргумент, който се предава на функцията `start_routine`.

Спиране на нишки

- Има няколко начина, по които една Pthread нишка може да бъде унищожена:
 - Функцията, която се изпълнява от нишката завършва нормално работата си и извиква **return**.
 - Функцията, която се изпълнява от нишката извиква библиотечната функция `pthread_exit()`.
 - Нишката може да бъде спряна от друга нишка като се използва библиотечната функция `pthread_cancel()`.
 - Когато процеса, в който живее нишката, бъде унищожен, се унищожават и всички негови нишки.

Спиране на нишки: `pthread_exit()`

```
void pthread_exit(void *value_ptr);
```

- Функцията `pthread_exit()` се използва за изрично спиране на нишка. Типично `pthread_exit()` се извиква след като нишката е завършила своята работа и повече не е необходима.
- Ако главната функция `main()` завърши работа преди нишките, които е създадала, то всички нишки също ще завършат работа. Когато обаче главната функция `main()` използва `pthread_exit()` за завършване на работа, останалите нишки могат да продължат да работят.
- Функцията `pthread_exit()` не затваря файловете, които са отворени в рамките на нишката.

Пример: pthread

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define NUM_THREADS 5
5
6 void *print_hello(void *thread_id){
7     int tid;
8     tid = (int)thread_id;
9     printf("Hello World! It's me, thread %d!\n", tid);
10    pthread_exit(NULL);
11 }
```


Пример: pthread

```
1 int main(int argc, char *argv[]){
2     pthread_t threads[NUM_THREADS];
3     for(int t=0;t<NUM_THREADS;t++){
4         printf("In main: creating thread %d\n", t);
5         int rc = pthread_create(&threads[t], NULL,
6                                 print_hello, (void *)t);
7         if (rc){
8             printf("ERROR; pthread_create() return %d\n",
9                 rc);
10            exit(-1);
11        }
12    }
13    printf("MAIN going to exit...\n");
14    pthread_exit(NULL);
15    return 0;
16 }
```

Пример: pthread

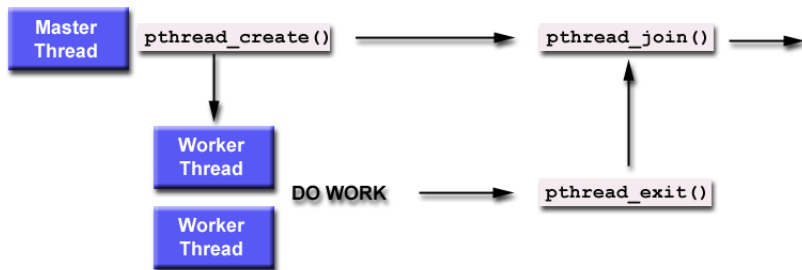
```
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #0!
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
In main: creating thread 4
MAIN going to exit...
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

Присъединяване на нишки

```
int pthread_join(pthread_t thread, void **value_ptr);
```

- Присъединяването е начин за синхронизиране между нишки. В библиотеката pthread за присъединяване се използва функцията pthread_join.
- Функцията pthread_join() блокира нишката, която я извиква, докато нишката с даден идентификатор не завърши работа.
- Всяка нишка може да бъде присъединена само веднъж. Опитът една нишка да бъде присъединена няколко пъти е логическа грешка.

Присъединяване на нишки



Фигура : <http://www.llnl.gov/computing/tutorials/pthreads/>

Пример: pthread

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define NUM_THREADS 3
6
7 void *busy_work(void *null){
8     double result=0.0;
9     for (int i=0; i<1000000; i++){
10         result = result + (double)random();
11     }
12     printf("result_□=□%e\n",result);
13     pthread_exit((void *) 0);
14 }
```

Пример: pthread

```
1 int main (int argc, char *argv[]){
2     pthread_t thread[NUM_THREADS];
3     for(int t=0; t<NUM_THREADS; t++){
4         printf("Creating □thread□%d\n", t);
5         int rc = pthread_create(&thread[t], NULL,
6             busy_work, NULL);
7         if (rc){
8             printf("ERROR; □pthread_create()□return□%d\n",
9                 rc);
10            exit(-1);
11        }
12    }
```

Пример: pthread

```
1  for(int t=0; t<NUM_THREADS; t++){
2      void *status;
3      int rc = pthread_join(thread[t],
4                          (void **)&status);
5      if (rc){
6          printf("ERROR; pthread_join() return %d\n",
7              rc);
8          exit(-1);
9      }
10     printf("Join thread %d status=%ld\n",
11         t, (long) status);
12 }
13 pthread_exit(NULL);
14 return 0;
15 }
```

Пример: pthread_join()

```
Creating thread 0  
Creating thread 1  
Creating thread 2  
result = 1.073687e+15  
result = 1.073788e+15  
Join thread 0 status= 0  
Join thread 1 status= 0  
result = 1.073502e+15  
Join thread 2 status= 0
```