# Средства и системы параллельного программирования
# Параллельное программирование для высокопроизводительных вычислительных систем

Семинар #1

Организационная информация. Основы многопоточных вычислений. Организация многопоточности. Pthreads и C++ threads(?)

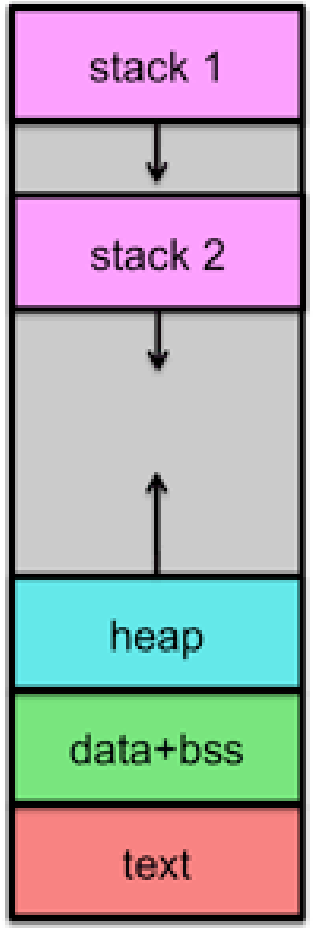
# Что такое поток исполнения?

# Что такое поток исполнения?

- A *thread of execution* is a sequence of instructions that can be executed concurrently with other such sequences in *multithreading* environments, while sharing a same address space.

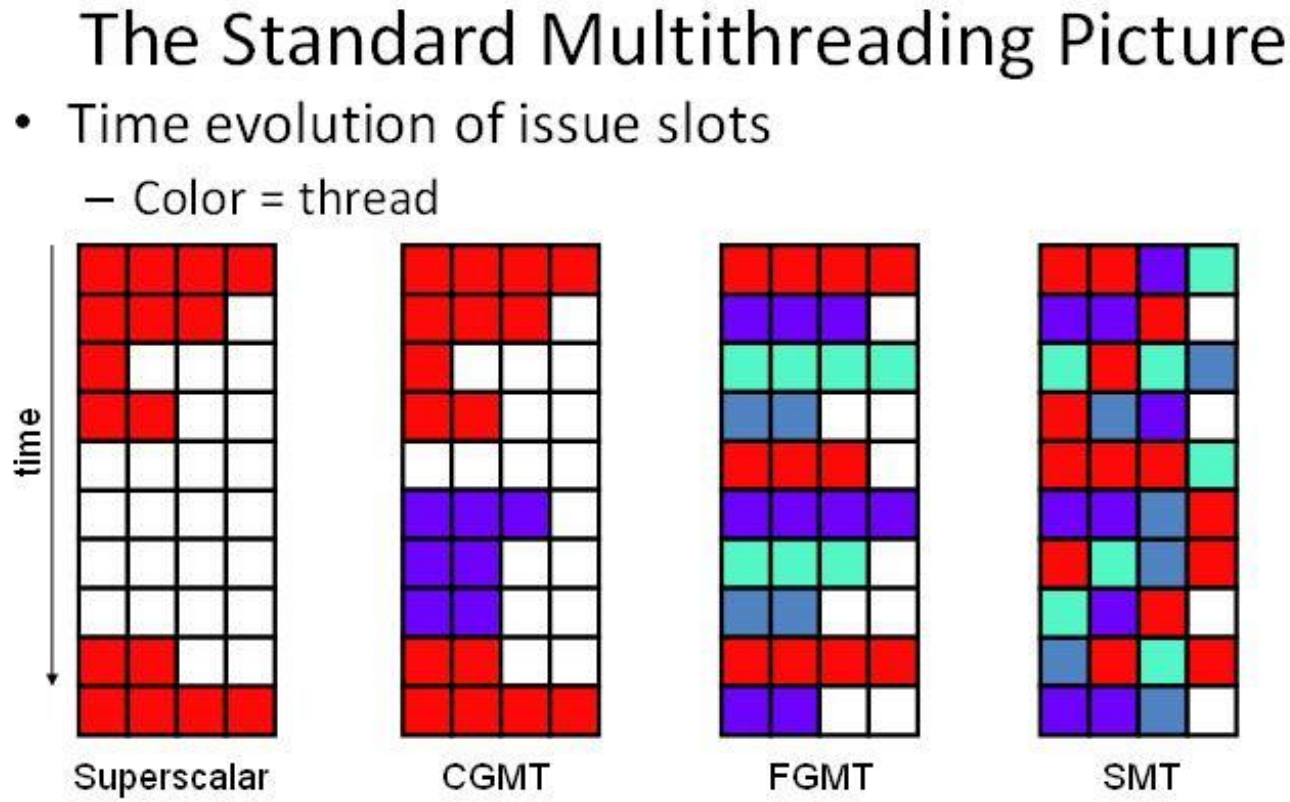
# Processes vs threads

| Per process items | Per thread items |
|---|---|
| Address space | Program counter |
| Global variables | Registers |
| Open files | Stack |
| Child processes | State |
| Signals and signal handlers | |
| Accounting information | |

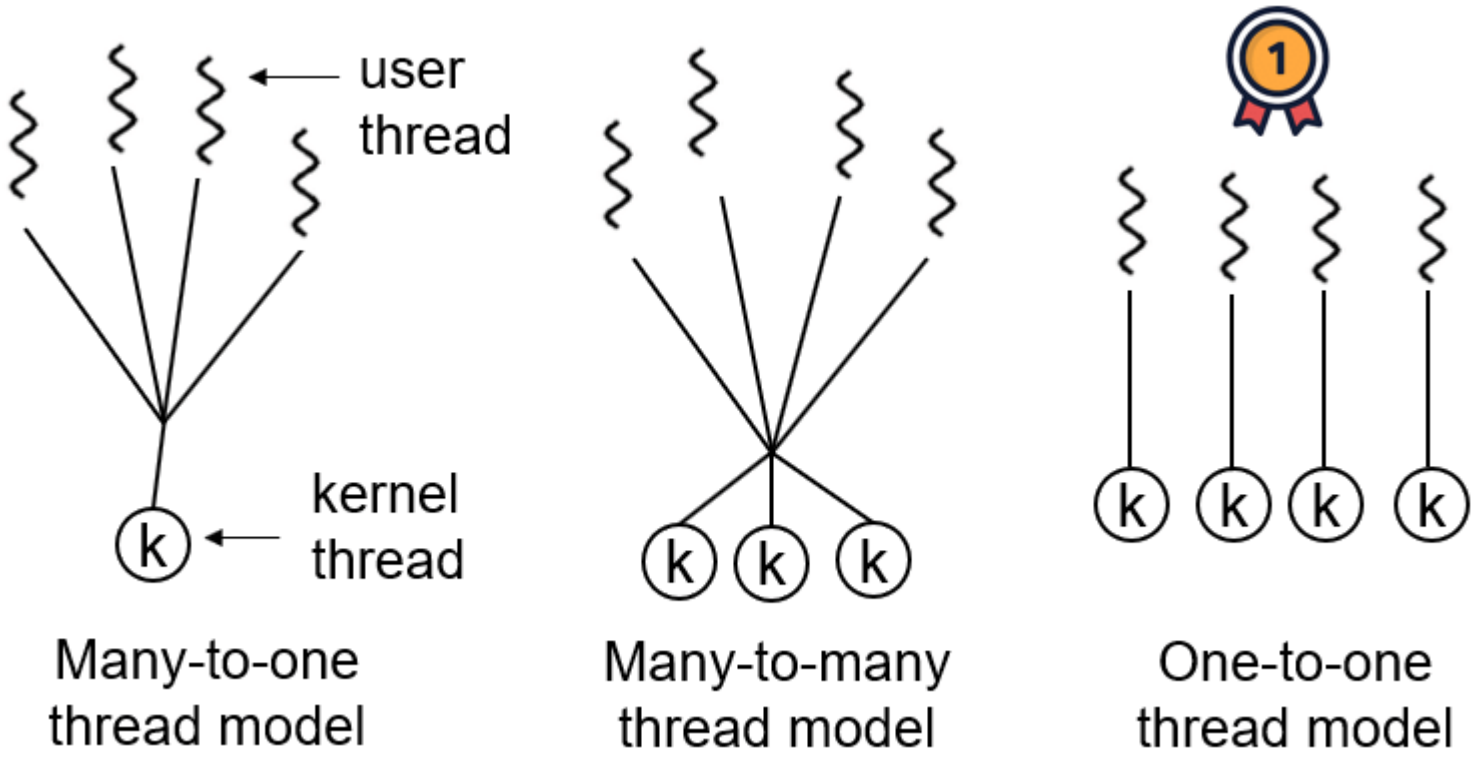


# Два вида многопоточности

- временнАя
- одновремЕнная



# Threading models




Many-to-one thread model

Many-to-many thread model

One-to-one thread model

# Fork() vs clone()

- `int clone(int (*fn)(void *_Nullable), void *stack, int flags, void *_Nullable arg, ... /* pid_t *_Nullable parent_tid, void *_Nullable tls, pid_t *_Nullable child_tid */ );`

- CLONE_VM (since Linux 2.0) (If CLONE_VM is set, the calling process and the child process run in the same memory space. If CLONE_VM is not set, the child process runs in a separate copy of the memory space of the calling process at the time of the clone call)
- CLONE_THREAD (since Linux 2.4.0) (If CLONE_THREAD is set, the child is placed in the same thread group as the calling process)

# Pthreads - создание и завершение потока

```
#include <pthread.h>


int pthread_create(pthread_t *restrict thread, const pthread_attr_t
*restrict attr, void *(*start_routine)(void *), void *restrict arg);

void pthread_exit(void *value_ptr);
```

# Pthreads – атрибуты потока

- `int pthread_attr_init(pthread_attr_t *attr);`
- `int pthread_attr_destroy(pthread_attr_t *attr);`

| Атрибут | Возможные значения | Описание |
|---|---|---|
| detachstate | PTHREAD_CREATE_DETACHED<br>PTHREAD_CREATE_JOINABLE | Определяет тип создаваемого потока: присоединяемый или отсоединенный |
| guardsize | Размер в байтах | Задает размер охранной зоны для защиты от переполнения стека |
| stackaddr | Адрес в байтах | Задает адрес стека |
| stacksize | Размер в байтах | Задает размер стека |

# Pthreads - взаимодействие с потоками

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

# Std::thread – создание потока

- thread() noexcept;
- thread( thread&& other ) noexcept;
- template< class F, class... Args >
  explicit thread( F&& f, Args&&... args );
- thread( const thread& ) = delete;

# Что может являться аргументом thread?

- Отдельная void-функция
- Функтор (класс с перегруженным оператором “()” )
- void Функция-член класса
- Лямбда-функция

# C++ threads vs pthreads

- В C++ thread нет аналога pthread_cancel()
- В C++ thread нельзя контролировать размер стека для каждого конкретного потока.
- В pthreads сложнее работать с исключениями и невозможно – с временами жизни объектов
- В C++ thread больше возможностей для написания функций для их исполнения в отдельном потоке

# Thread affinity

- #define _GNU_SOURCE /* See feature_test_macros(7) */
- #include <sched.h>

// для потока с номером pid_t выставить битовую маску, равную mask

- int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
- // для потока с номером pid_t узнать битовую маску и поместить её в mask
- int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);

# Thread affinity

```
void CPU_ZERO(cpu_set_t *set); // обнулить маску

void CPU_SET(int cpu, cpu_set_t *set); //добавить CPU #cpu к набору
void CPU_CLR(int cpu, cpu_set_t *set); // убрать CPU #cpu из набора
int  CPU_ISSET(int cpu, cpu_set_t *set); // определить, используем ли CPU #cpu в наборе

int  CPU_COUNT(cpu_set_t *set); //число используемых CPU в наборе
```