Средства и системы параллельного программирования

Семинар #1. Введение в многопоточность. Потоки в Linux. POSIX threads

Что такое поток?

Поток (исполнения) — наименьшая последовательность программных инструкций, которая может управляться планировщиком операционной системы

Выполнение программы состоит из выполнения всех её потоков

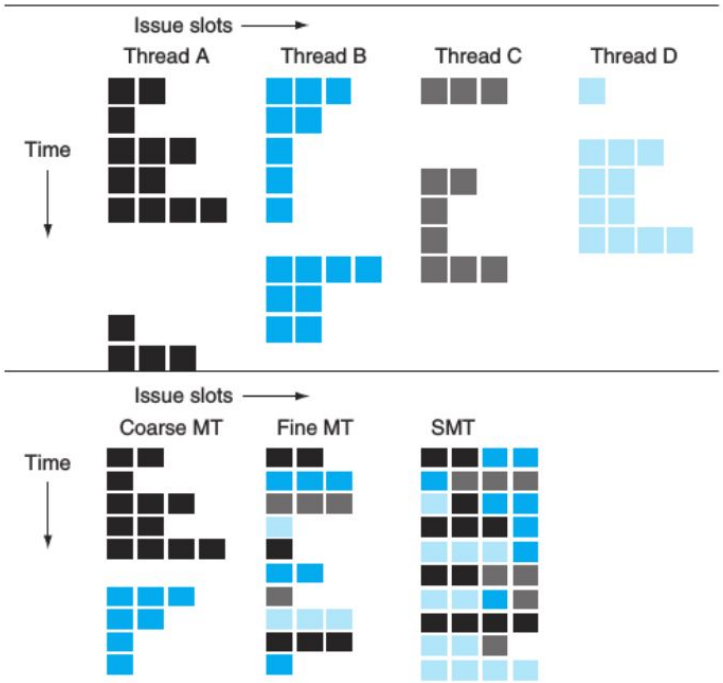
При этом **процесс** - изолированная среда выполнения и контейнер ресурсов, в пределах которой выполняется один или несколько **потоков**.

Аппаратная и логическая многопоточность

Аппаратная многопоточность (**parallelism**) - свойство программы и (прежде всего) системы выполнять различные потоки исполнения на различных функциональных устройствах

Логическая многопоточность (**concurrency**) - свойство программы порождать и ставить на исполнение функциональными устройствами более одного потока управления. Реальное параллельное выполнение **не гарантируется**

Виды многопоточности в рамках одного ядра

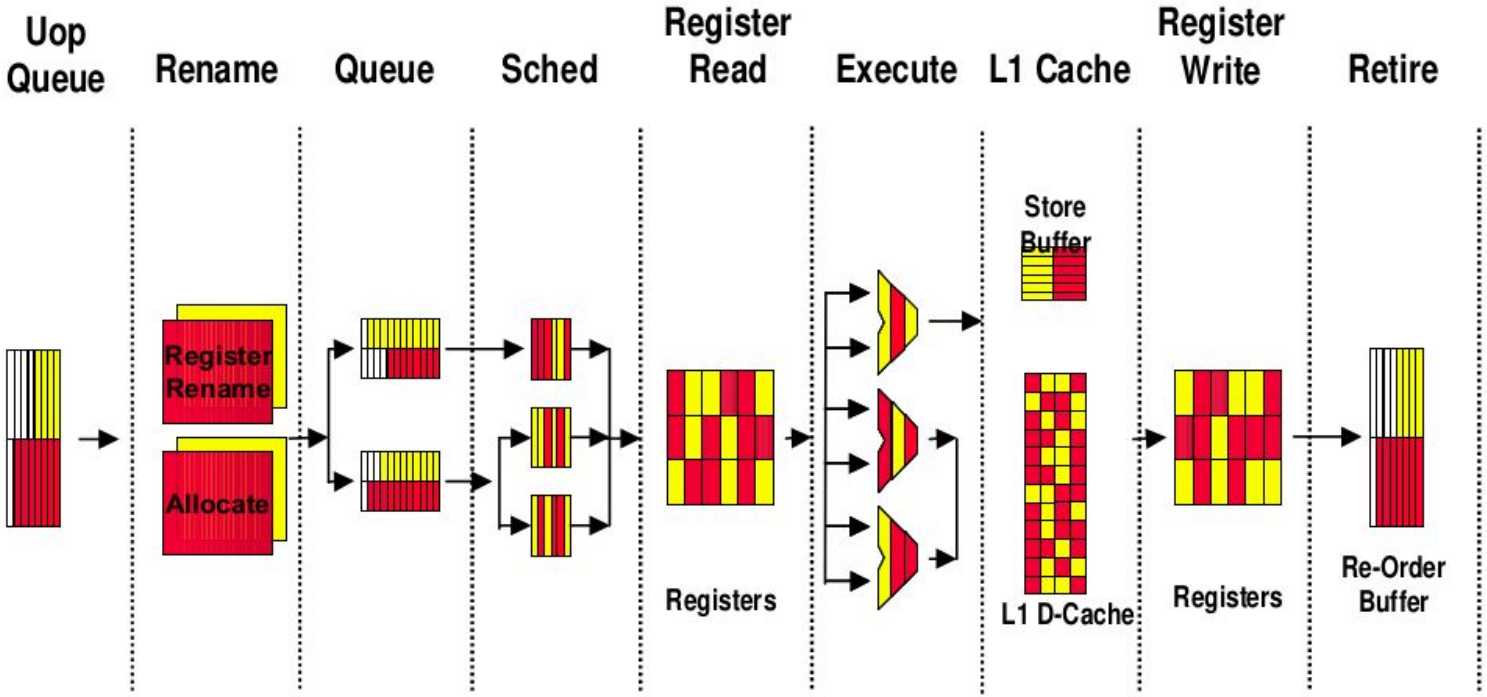


ВременнАя (*temporal*) многопоточность - переключение контекста потока связано с квантами времени. В зависимости от частоты смены различают Coarse и Fine стратегии

Одновременная (*simultaneous*) многопоточность - возможность использовать в рамках одного такта инструкции из разных потоков

(см. lscpu - Thread(s) per core)

Intel hyper-threading



https://habr.com/ru/companies/ruvds/articles/836978/

Процесс и поток - в чём отличие?

Сущности, общие для всех потоков	Сущности, приватные для каждого потока
Адресное пространство	Program counter
Глобальные переменные программы	Значения регистров
Открытые файлы (дескрипторы)	Стек
Дочерние процессы	Thread-local storage
Сигналы и обработчики сигналов	TID потока
PID процесса	

А с точки зрения ОС Linux?

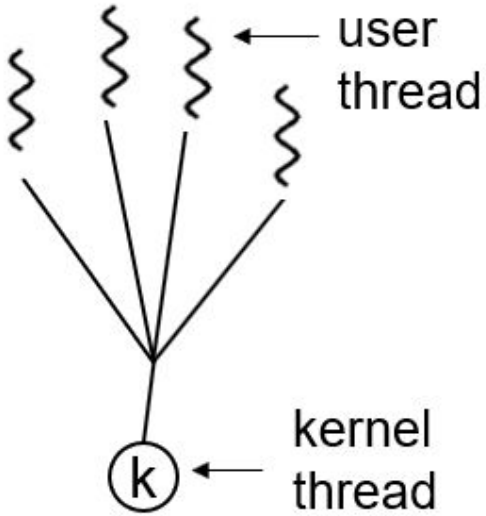
С точки зрения ядра ОС Linux, отличия между процессом и потоком нет. Более точно, ядро оперирует планируемыми объектами, а процесс не планируется сам по себе.

Планируемый объект в ядре Linux описывается структурой **task_struct** (https://elixir.bootlin.com/linux/v6.16.5/C/ident/task_struct)

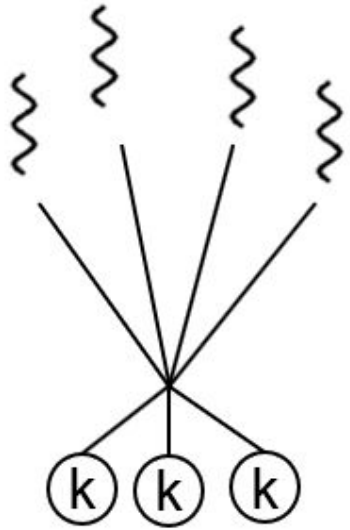
Многие значения task_struct можно посмотреть напрямую в /proc/PID/task/TID

Вывести информацию о всех работающих task: ps -eLf

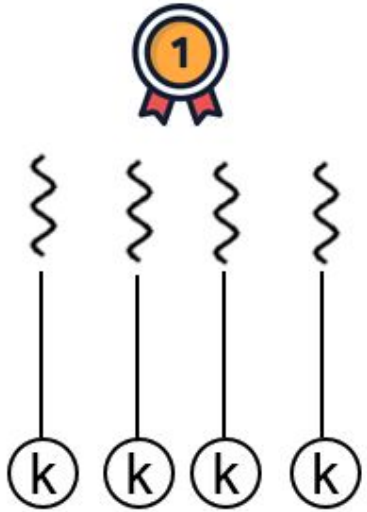
Отображение user-space потоков в kernel-space потоки



Many-to-one
thread model



Many-to-many
thread model



One-to-one
thread model

Системный вызов clone

```
#define _GNU_SOURCE

#include <sched.h>

int clone(int (*fn)(void *), void *child_stack, int flags, void *arg, ...);
```

Обёртка для системного вызова clone, цель которого - создать новую сущность (процесс, поток), используя более тонкие настройки разделения ресурсов, чем в fork()

fn - указатель на функцию, выполняемую процессом

child_stack - указатель на будущий стек дочернего процесса

arg - аргументы, передаваемые в fn

flags - настройки разделения ресурсов. От них зависит, появится у нас процесс или поток!

Базовые актуальные флаги: **CLONE_VM** (оба "процесса" используют единое адресное пространство)

CLONE_THREAD (создаваемый "процесс" попадает в ту же thread group, что и вызываемый)

Pthreads

Pthreads, они же POSIX threads - решение по организации многопоточных вычислений, переносимое для всех платформ, поддерживающих стандарт POSIX.

Идентификатор потока является значением типа pthread_t, причём реальный тип pthread может быть различным

Для использования pthreads в приложениях необходимо подключить
#include <pthread.h>

Функция создания потока в pthread

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

thread - указатель на идентификатор потока

attr - атрибуты потока

start_routine - указатель на функцию, исполняемую потоком

arg - аргументы для start_routine

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 PTHREAD_CREATE_JOINABLE (по умолчанию)	Определяет тип создаваемого потока: присоединяемый или отсоединенный
guardstate	Размер в байтах	Задает размер охранной зоны для защиты от переполнения стека
stackaddr	Адрес	Задает адрес стека
stacksize	Размер в байтах	Задает размер стека

Множество функций для установки полей атрибутов

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int
detachstate);

int pthread_attr_setguardsize(pthread_attr_t *attr, size_t
guardsize);

int pthread_attr_setstack(pthread_attr_t *attr, void
*stackaddr, size_t stacksize);

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t
stacksize);
```

Виды потоков при их исполнении

Joinable - поток, по окончании работы требующий присоединения к другому потоку для получения результатов выполнения и освобождения ресурсов

Detached - поток, ресурсы которого освобождаются автоматически после выполнения последней инструкции. Результат вернуть нельзя

Вид потока может быть выставлен с помощью атрибутов, а также с помощью функции `int pthread_detach(pthread_t thread);`

Синхронизация потоков и работа с результатом

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

Функция присоединения потока

thread - идентификатор потока, который мы хотим присоединить

value_ptr - область памяти, куда поток с идентификатором thread кладёт результат

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

Функция, вызываемая потоком для (само)завершения с передачей результата в присоединяющий поток

value_ptr - указатель на возвращаемые данные

Привязка потоков к ядрам

```
#define _GNU_SOURCE
```

int pthread_setaffinity_np(pthread_t thread, size_t cpusetsize,

const cpu_set_t *cpuset);

thread - поток, привязку для которого мы хотим задать

cpusetsize - длина массива cpuset

cpuset - маска для явного задания номеров CPU, участвующих при привязке потоков

https://linux.die.net/man/3/cpu_set

Задание

Реализовать многопоточный подсчёт гистограммы по массиву элементов. Исходные данные: массив из чисел типа uint16_t длиной N. Число бинов в гистограмме = 256.

- Разбить диапазон [0, N) на T неперекрывающихся сегменты. N и T должны быть заданы в командной строке.
- Каждый поток считает собственную локальную гистограмму по своему сегменту.
- После **join** главный поток выполняет редукцию для подсчёта итоговой гистограммы.
- Многопоточность необходимо реализовать средствами библиотеки pthreads. Не разрешается использовать средства синхронизации потоков, отличные от **join**.
- Исходный массив необходимо инициализировать псевдослучайными числами в главном потоке до начала работы параллельных потоков

Требования к решению/отчёту:

- Проведение проверки корректности: сумма всех бинов глобальной гистограммы равна N.
- Отчётность по производительности: измерить время однопоточной и многопоточной версий, посчитать ускорение и эффективность распараллеливания для следующего числа потоков: {1, 2, 4, 8, 12, 16}.
- Для подсчёта показателей из предыдущего пункта рассматривайте большие N (>10^9)
- Сформировать небольшой отчёт, добавив информацию про используемую для выполнения задания аппаратную платформу

Дедлайн: 22.09, 29.09

Сдача заданий

Задание можно сдавать, присылая исходный код и отчёт на почту dimlichmanov@gmail.com

Но предпочтительнее вариант с Github. Необходимо:

- Создать приватный репозиторий на Github
- Добавить меня (dimlichmanov) в collaborators в настройках репозитория
- Каждое новое задание делать в новой рабочей ветке, отличной от master
- По готовности задания создать pull request из рабочей ветки в master, поставить меня в Reviewers в меню Pull request
- Нажать create pull request и ждать обратной связи. Уведомления о моих комментариях придут вам на почту, привязанную к github
- В случае моего approve ваших изменений, слить ветку в master

Материалы по курсу находятся в репозитории dimlichmanov/tspp_2025