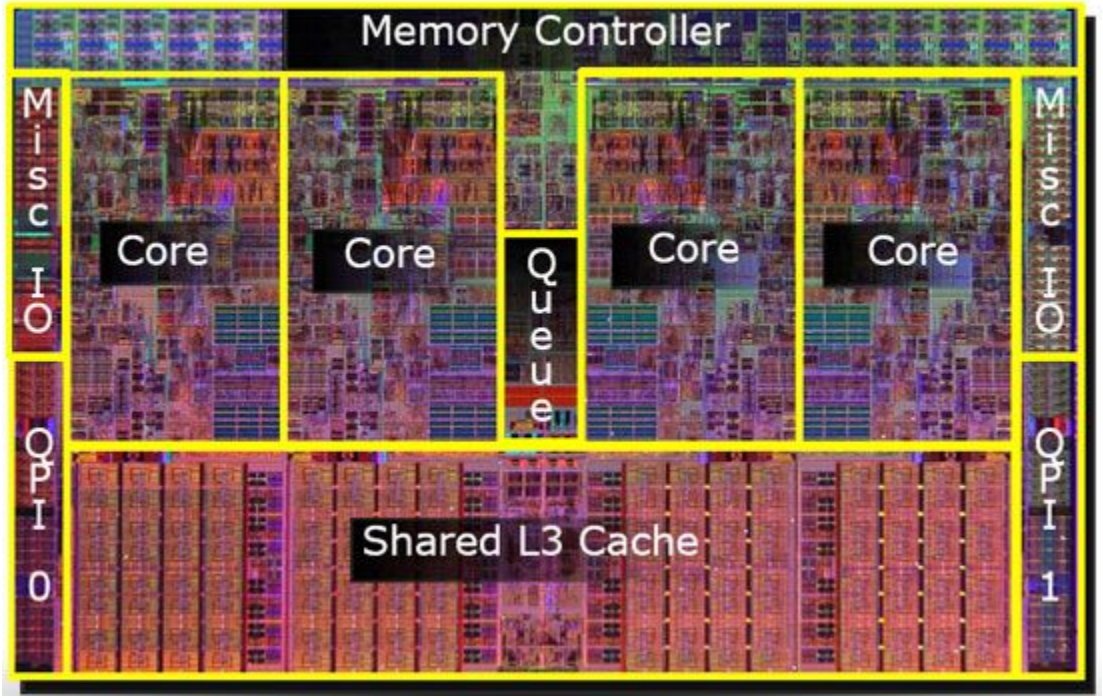
Многопоточное программирование на C++

Потоки и примитивы синхронизации

Модель памяти

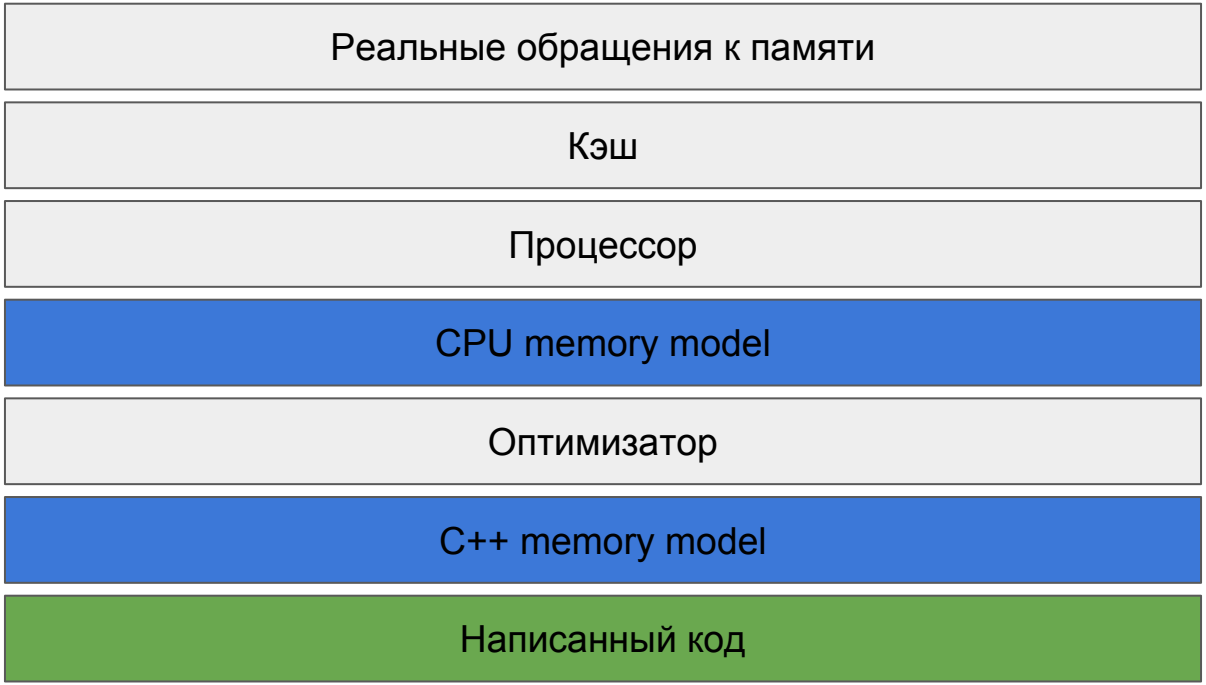
Сколько стоит обращение к памяти



Исполнение кода

Исполнение кода

Интерфейс между абстракциями



Контракт

OBJ

Developer

Compiler

CPU

Контракт

При отсутствии **race condition** исполнение кода выглядит так же, как он написан

Контракт

При отсутствии **race condition** исполнение кода выглядит так же, как он написан

- Чтение одного **memory location** из разных потоков - ОК
- Запись разных **memory location** из разных потоков - ОК
- Запись одного **memory location** из разных потоков - **race**
- Запись/Чтение одного **memory location** из разных потоков - **race**

Контракт

При отсутствии **race condition** исполнение кода выглядит так же, как он написан

Race condition

- Чтение одного **memory location** из разных потоков - ОК
- Запись разных **memory location** из разных потоков - ОК
- Запись одного **memory location** из разных потоков - **race**
- Запись/Чтение одного **memory location** из разных потоков - **race**

Memory location

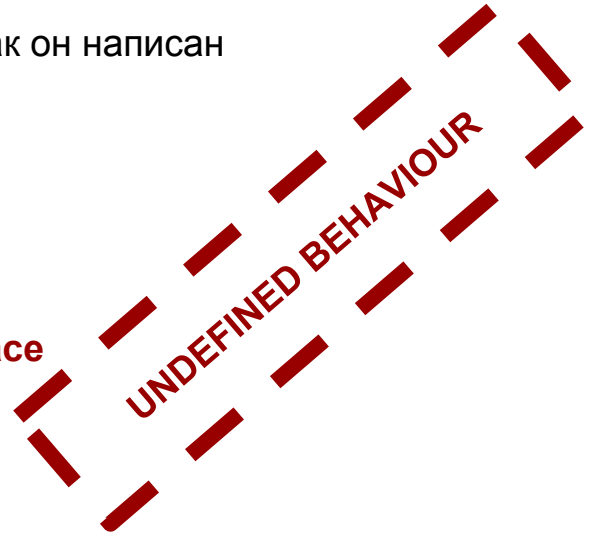
- Каждый объект в C++ может занимать 1 или более **memory location**
- Простые типы - ровно 1 **memory location**
- Смежные битовые поля - 1 **memory location**

Контракт

При отсутствии **race condition** исполнение кода выглядит так же, как он написан

Race condition

- Чтение одного **memory location** из разных потоков - ОК
- Запись разных **memory location** из разных потоков - ОК
- Запись одного **memory location** из разных потоков - **race**
- Запись/Чтение одного **memory location** из разных потоков - **race**



Memory location

- Каждый объект в C++ может занимать 1 или более **memory location**
- Простые типы - ровно 1 **memory location**
- Смежные битовые поля - 1 **memory location**

Потоки

posix_threads

(до ядра 2.6) Linux Threads

\+ Использовали *clone* для создания процессов с разделяемой памятью, не требовали специальной поддержки в ядре

\- Библиотека не могла выполнить все требования стандарта POSIX

(начиная с ядра 2.6) NTPL - Native Linux Threads Library, NGPT - Next Generation Posix Threads (close in 2003)

\+ Использует поддержку ядра

\+ Полностью совместима со стандартом

\+ Является частью libc в Linux

man pthreads

*Очень много **крайне** полезных букв…….*

getconf GNU_LIBPTHREAD_VERSION

Какая библиотека используется в вашей системе

std::thread

Позволяет запустить новый поток исполнения

- native_handle_type, платформа зависимый дескриптор потока. Бывает полезен, когда нужна функция не представленная в стандартной библиотеке
    - native_handler() - возвращает дескриптор
- join() - блокирует вызывающий поток до тех пор, пока вызываемый поток не закончит исполнение
- joinable() - возвращает true если у потока можно вызвать метод join
- detach() - отвязывает поток от объекта std::thread, при завершении **потока (не объекта)** все ресурсы будут освобождены

Поток запускается автоматически при создание объекта std::thread:

- ...Кроме специального случая: конструктор без аргументов создает объект без создания/запуска потока
- `template< class Function, class... Args >`
  `explicit thread( Function&& f, Args&&... args );`

Объекты потока нельзя копировать!

Синхронизация

Семафоры - база всех примитивов

```
init(n):
    счётчик := n


enter():
    счётчик := счётчик - 1


leave():
    счётчик := счётчик + 1
```

man sem_overview

- sem_close
- sem_destroy
- sem_getvalue
- sem_init
- sem_open
- sem_post
- sem_unlink
- sem_wait
- pthreads
- shm_overview

Futux - поддержка ядра для синронизации

доступно начиная с ядра 2.6

В основе это просто кусок разделяемой памяти, обычно число, выровненное по машинному слову + очередь ожидания в ядре

Доступ осуществляется с **atomic** инструкция процессора

В случае, если 2 процесса не могут договориться о значении или процесс вынужден ждать , управление отдается ядру

man 7 futex

Полезное чтение

std::mutex - c++11 синхронизация

Бинарный семафор, позволяет сделать только lock/unlock

- std::mutex
  Простой mutex, нельзя захватывать рекурсивно

- std::timed_mutex
  Можно захватить на какое-то время

- std::recursive_mutex
  Можно захватывать рекурсивно

- std::recursive_timed_mutex
  Композиция всех типов

std::*lock* - RAII обертка для блокировок

Обертка для безопасного захвата и освобождения mutex’ов

- std::lock_guard
  Просто делает lock/unlock в текущем блоке

- std::unique_lock
  Захватывает/отпускает mutex, но поддерживает все возможные опции, такие как захват на время, не захватывать лок, но безопастно его отпустить, и.т.д

std::condition_variable - управление ожиданием

Позволяет дождаться определенного события, случившегося в другом потоке

- std::condition_variable
  - wait
  - wait_for
  - wait_until
  - notify_one
  - notify_all

Код

Ссылки

- Проект: https://github.com/xphoenix/afina
  - cmake
  - ccahce (optional)
  - Make / ninja
  - gcc (!)
- Домашки
  - На вики проекта
  - Не забывайте напоминать обновить
  -

Присылать код на ph.andronov@corp.mail.ru

Сборка

```
user@host ~] git clone git@github.com:<your-user-name>/afina.git
Cloning into 'afina'...
user@host afina] cd afina
user@host afina] git remote add upstream git@github.com:xphoenix/afina.git
user@host build] mkdir build && cd build
user@host build] cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_EXPORT_COMPILE_COMMANDS=y
-DECM_ENABLE_SANITIZERS="memory,address" ..
-- The C compiler identification is GNU 8.2.1
-- The CXX compiler identification is GNU 8.2.1
-- Check for working C compiler: /usr/bin/cc
...
user@host build] make
…
user@host build] ./src/afina
```