Информатика. Семинар №10

26.04.2016

https://dl.dropboxusercontent.com/u/96739039/sem4/infa_s10.pdf

Многопоточность в C++11

```
#include·<iostream>·
#include·<thread>
#include·<mutex>

int·main()·{
··int·x·=·0;

··std::thread·t0([&]()·{
····for·(int·i·=·0;·i·<·1000000;·++i)·x++;
··});

··std::thread·t1([&]()·{
····for·(int·i·=·0;·i·<·1000000;·++i)·x--;
··});

··t0.join();·t1.join();

··std::cout·<<·x·<<·std::endl;
··return·0;
}
```

P.S. Форматирование кода определяется нехваткой места на слайде

join vs detach

- https://habrahabr.ru/post/182610/
- ***join*** дожидается завершения работы нити
- ***detach*** позволяет отсоединить поток от объекта, иными словами, сделать его фоновым (полезно, если мы запускаем нить в отдельной ф-и, а нить создана на стеке, т.е. должна быть удалена при выходе из неё)
- Для нити обязательно надо вызвать либо join, либо detach

std::mutex

```
int main() {
  int x = 0;
  std::mutex m;

  std::thread t0([&]() {
    for (int i = 0; i < 1000000; ++i) {
      m.lock(); x++; m.unlock();
    }
  });

  std::thread t1([&]() {
    for (int i = 0; i < 1000000; ++i) {
      m.lock(); x--; m.unlock();
    }
  });

  t0.join(); t1.join();

  std::cout << x << std::endl;
  return 0;
}
```

Самый простой пример dead-lock`a

```
int main() {
  std::mutex m0, m1;

  std::thread t0([&]() {
    for (int i = 0; i < 10000; ++i) {
      m0.lock(); m1.lock(); m0.unlock(); m1.unlock();
    }
  });

  std::thread t1([&]() {
    for (int i = 0; i < 10000; ++i) {
      m1.lock(); m0.lock(); m0.unlock(); m1.unlock();
    }
  });

  t0.join(); t1.join();
  std::cout << "Done";
  return 0;
}
```

std::lock_guard

```
int main() {
  int x = 0;
  std::mutex m;

  std::thread t0([&]() {
    for (int i = 0; i < 1000000; ++i) {
      std::lock_guard<std::mutex> g(m);
      x++;
    }
  });

  std::thread t1([&]() {
    for (int i = 0; i < 1000000; ++i) {
      std::lock_guard<std::mutex> g(m);
      x--;
    }
  });

  t0.join(); t1.join();
  std::cout << x << std::endl;
  return 0;
}
```

А если измерить время работы?

```
#include·<iostream>·
#include·<thread>
#include·<mutex>
#include·<chrono>

int·main()·{
··int·x·=·0;·const·int·N·=·100000000;·std::mutex·m;

··typedef·std::chrono::high_resolution_clock·clock;
··clock::time_point·start·=·clock::now();

··std::thread·t0([&]()·{
····for·(int·i·=·0;·i·<·N;·++i)·{·std::lock_guard<std::mutex>·g(m);·x++;·}
··});

··std::thread·t1([&]()·{
····for·(int·i·=·0;·i·<·N;·++i)·{·std::lock_guard<std::mutex>·g(m);·x--;}
··});

··t0.join();·t1.join();

··std::chrono::duration<double>·duration·=·clock::now()·-·start;
··std::cout·<<·x·<<·"·"·<<·duration.count()·<<·"s"·<<·std::endl;
··return·0;
}
```

Код с синхронизацией с помощью мьютексов работает в десятки-сотни раз дольше

std::atomic

Для типов с тривиальным конструктором копирования (все скаляры / в случае класса сами явно не переопределяли его) можно использовать std::atomic. Может работать в разы-десятки раз быстрее std::mutex.

```
#include <atomic>

int main() {
  std::atomic<int> x = 0; const int N = 100000000;

  typedef std::chrono::high_resolution_clock clock;
  clock::time_point start = clock::now();

  std::thread t0([&]() {
    for (int i = 0; i < N; ++i) x++;
  });
```

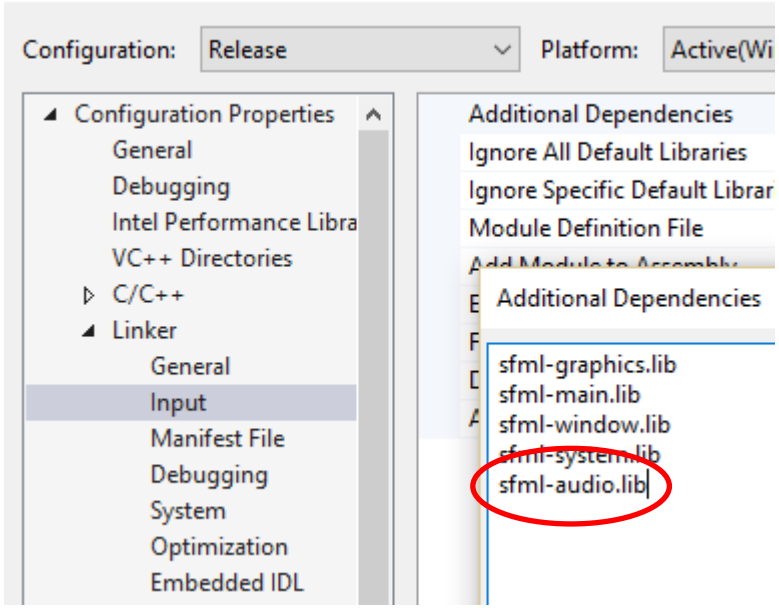
Lock-free programming: про что это? (compare and swap)

- http://preshing.com/20120612/an-introduction-to-lock-free-programming/
- http://users.livejournal.com/_foreseer/34284.html
- http://kukuruku.co/hub/cpp/lock-free-data-structures-the-evolution-of-a-stack
- https://habrahabr.ru/post/195770/

P.S. Если вы в проекте собираете данные из нескольких нитей в одну структуру, то желательно разобраться и протестировать производительность в сравнении с mutex-based structures.

Работа со звуком в sfml

- Говорим, что необходимо слинковаться с соответствующей библиотекой (для Debug и Release отдельно).



Работа со звуком в sfml

- http://www.sfml-dev.org/tutorials/2.3/audio-sounds.php

```
#include <SFML\Audio.hpp>

int main()
{
    sf::Music music;
    if (!music.openFromFile("music.ogg"))
        return -1; // error
    music.play();

    while (window.isOpen())
    {
        // ...
    }
}
```

P.S. SFML supports the audio file formats WAV, OGG/Vorbis and FLAC. Due to licensing issues MP3 is **not** supported.

P.P.S. sf::Music и sf::Sound проигрываются в отдельных потоках + могут автоматически микшироваться

Упражнение 1

- По циклу должно проигрываться несколько файлов, заданных в массиве строк:

```
std::vector<std::string> files({"file0.wav", "file1.wav" , "file2.wav"});
```

```
class MusicPlayer {
public:
  void SetMusic(const std::vector<std::string>& musicFiles) {
    // TODO
  }
  void Play() {
    // TODO: проигрывание музыки должно происходить в отдельном потоке
  }
};
```

Звуки

- Звуки – небольшие файлы, которые вызываются части и помещаются в оперативную память

```
#include <SFML/Audio.hpp>

int main()
{
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("sound.wav"))
        return -1;

    ...

    return 0;
}
```

Звуки

- Можно генерировать программно

```
std::vector<sf::Int16> samples = ...;
buffer.loadFromSamples(&samples[0], samples.size(), 2, 44100);
```

Since ***loadFromSamples*** loads a raw array of samples rather than an audio file, it requires additional arguments in order to have a complete description of the sound. The first one (third argument) is the number of channels; 1 channel defines a mono sound, 2 channels define a stereo sound, etc. The second additional attribute (fourth argument) is the sample rate; it defines how many samples must be played per second in order to reconstruct the original sound.

P.S. &samples[0] – указатель на начало данных (лежат последовательно). Запись равносильна samples.data()

sf::Sound

- Now that the audio data is loaded, we can play it with a sf::Sound instance.

```
sf::SoundBuffer buffer;
// load something into the sound buffer...

sf::Sound sound;
sound.setBuffer(buffer);
sound.play();
```

Упражнение 2 («синтезатор»)

- Для каждой клавиши своя частота «гудения» (ля == 440 Гц)

Полная стандартная шкала частот музыкальных тонов

Нота	Частота, Гц								
	Суб-контр-октава	Контр-октава	Большая октава	Малая октава	Первая октава	Вторая октава	Третья октава	Четвертая октава	Пятая октава
До (B)		32,70	65,41	130,82	261,63	523,25	1046,5	2093,0	4186,0
До-диез ($C^{\#}$)		34,65	69,30	138,59	277,18	554,36	1108,7	2217,4	4434,8
Ре (D)		36,95	73,91	147,83	293,66	587,32	1174,6	2349,2	4698,4
Ре-диез ($D^{\#}$)		38,88	77,78	155,56	311,13	622,26	1244,5	2489,0	4978,0
Ми (E)	20,61	41,21	82,41	164,81	329,63	659,26	1318,5	2637,0	5274,0
Фа (F)	21,82	43,65	87,31	174,62	349,23	698,46	1396,9	2793,8	
Фа-диез ($F^{\#}$)	23,12	46,25	92,50	185,00	369,99	739,98	1480,0	2960,0	
Соль (G)	24,50	49,00	98,00	196,00	392,00	784,00	1568,0	3136,0	
Соль-диез ($G^{\#}$)	25,95	51,90	103,80	207,60	415,30	830,60	1661,2	3332,4	
Ля (A)	27,50	55,00	110,00	220,00	440,00	880,00	1720,0	3440,0	
Ля-диез (B)	29,13	58,26	116,54	233,08	466,16	932,32	1864,6	3729,2	
Си (H)	30,87	61,74	123,48	246,96	493,88	987,75	1975,5	3951,0	

Упражнение 2 («синтезатор»)

- Длительность гудения клавиши фиксирована (например, 0.5 сек)
- Громкость гудения плавно уменьшается (например, с помощью метода ***setVolume***)
- Звуки от разных клавиш могут звучать одновременно (mixing)

P.S. Можно программно сгенерировать затухающее Ля, а остальные частоты создать на её основе с помощью ***setPitch*** (Changing the pitch has a side effect: it impacts the playing speed)

Запись с микрофона

- http://www.sfml-dev.org/tutorials/2.3/audio-recording.php