Контрольные вопросы №7
Аляев Роман

Что такое контекстное переключение задач?

Контекстный переключатель– это переключение центрального процессора с одного процесса или потока на другой.
На семинаре мы рассмотрели псевдо-параллелизм, который как раз работает (создает иллюзию параллельности) за счет очереди задач у какого-либо ядра.

Назовите основные подходы к организации параллелизма?

Есть два основных подхода – это multi-processes и multi-threading.
Первый работает таким образом, что у каждого процесса свои данные и процессы выполняются за счет взаимодействия с единым блоком IPC – Inter Process Communication.
Второй же устроен таким образом, что данные общие и от них параллельно идут функции.

На тот случай, если я неправильно понял вопрос:
Есть два принципа распараллеливания программ – по данным (один большой массив данных и мы разбиваем его на кусочки, и каждый кусочек работает в своем потоке) и по задачам (каждый поток делает свою задачу обладая своими данными)

Что может влиять на производительность параллельных алгоритмов?

На последнем семинаре мы рассмотрели, что такое закон Амдала. Это некая зависимость, которая позволяет определить ускорение работы при добавлении бОльшего числа ядер. И как мы выяснили, не всегда больше == лучше. Есть некоторый лимит – если мы попытаемся его преодолеть добавлением ядер, то скорость работы уменьшится, потому что ядра также взаимодействуют между собой и передают друг другу информацию, а значит мы можем на простых задачах терять эффективность на бессмысленном общении ядер.

Как в стандартной библиотеке реализована концепция асинхронного исполнения?

Стандартная библиотека предлагает механизм future: статический объект, который предоставляет значение, но позже, когда нам это нужно. Есть три типа:

- async – самый слабый контроль за результатом
- packaged_task – контроль посильнее
- promise – самый сильный контроль

```
std::future<int> result = std::async( [](){return 42;}; ) // самый
слабый контроль
X = result.get();

std::packeged_taks<int()> task( [](){return 42;}; ); // контроль
посильнее
```

```
auto result = task.get_future();
std::tread(std::move(task).detach());
X = result.get();

std::promise<int> p; // самый сильный контроль
auto result = p.get_future();
std::thread([](std::promise<int> & p){p.set_value(42);},
std::ref(p).detach());
X = result.get();
```

То есть пока нам значение не нужно, программа его считает, а мы занимаемся своими делами; когда же нам понадобится это значение, программа либо его нам выдаст, либо мы подождем пока программа его досчитает и потом уже выдаст.

Что нужно учитывать при замене последовательной реализации алгоритма на параллельную?

При построении параллельных алгоритмов необходимо учитывать, что время доступа к данным может различаться. Для обеспечения эффективности алгоритма в этом случае следует в явном виде планировать распределение данных и схему обмена данными между потоками.