№1

Это процесс записи и восстановления состояния процесса или потока таким образом, чтобы в дальнейшем продолжить его выполнение с прерванного места.

№2

- multi-proccessing

несколько процессов работают над одной задачей (mp)

процесс оснащен всем, чем можно (код, данные, ресурсы)

у каждого процеса свои данные.

- multi-threading

работаем с процессом, в котором некоторые участки кода

работают независимо

потоки могут между собой взаимодействовать

у процессов общие данные

- принципы распараллеливания:

* по данным (разбиваем массив по кусочкам, потом объединяем результаты)
* по задачам (один сортирует, второй музыку играет, третий еще что-то)

№3

На неё влияют:

* кол-во ядер
* безопасность относительно исключений
* конкуренция за данные
* Закон Амдала

№4

future -- статический объект, который предоставляет значение с помощью get

инструменты для работы с ним:

- async

запускает в потоке функцию, результат завернет в объект типа future

std::future<int> result = std::async([](){return 42;});

- packeged\_task - учитывает информацию в потоке

std::packeged\_task<int()> task([]()/\*lambda,functor\*/{return 42;});

auto result = task.get\_future();

std::thread(std::move(task)).detach();

...

X = result.get();

- promice

можно работать, будто это известное значение

atd::promise<int> p;

auto result = p.get\_future;

std::thred([](std::promise<int>&p){})

...

X = result.get();

№5

Нужно учитывать, что переход на параллельную организацию не гарантирует стоящего повышения производительности. Напротив, иногда такой переход может привести к увеличению времени работы программы. Более того, использование такой организации может порождать конфликты в программе и ухудшать читаемость кода.