EPam *2016 Dec 15*

When will you choose multiprocessing, when will you choose multithreading?
Answer : When we need to kill one process, while keeping other running, we cannot use multithreading.
Answer : Multiprocessing allows failure/crash in particular tasks without affecting the others.

When working with multithreading, what problem we may encounter? How to solve?
Answer : Racing condition, solved by protection of critical session, using mutex lock, spinlock or lockless algo.

What is deadlock? Why ordering of locks can help to solve?
Answer : When a thread succeed in locking a resource, we are sure that all locks having lower precedences aren't lcoked.

Can windows function `waitformultipleobjects` help to solve deadlock? How is it implemented?
Answer : Yes, `waitformultipleobjects(A,B,C)`, where A, B, C are locks, because it waits the objects in order as the input.

Is there any synchronization tools provided in C++11?
Answer : Yes, future, promise, packaged task, async, conditional variable.

Is there timed mutex, what is it?
Answer : Yes, it is a mutex with extra function : try-lock-for and try-lock-until.

Tell me something about C++11 atomic?
Answer : Atomic is a template class, ensuring that its operations are uninterrupted before completion.
Answer : It can be lock-free, but not necessary.

What is `std::vector<T>::emplace_back`?
Answer : It is a move-semantics version of `push_back`.

```
void std::vector<T>::push_back(const T& arg);
void std::vector<T>::emplace_back(T&& arg);
```

What is move semantics?
Answer : It's a swap with a temporary object, without copying. It works for rvalue object passed by rvalue reference only.

Tell me something about memory management?
Answer : When we do dynamic allocation, there may be memory leakage when we forget to delete or exception throws.
Answer : Therefore we need smart pointers.

Can we replace all unique pointer with shared pointer? Why do we still need unique pointer?
Answer : Use shared pointer only when shared ownership is necessary, as unique pointer is faster.
Answer : There is no reference count, no manager in unique pointer.

What is weak pointer?
Answer : It is an observer of shared pointer, it is used when shared pointers form a loop.

Why do we need smart pointers? Can we replace it with raw pointers provided that we can handle delete properly?
Answer : It is exception safe.

Name different exception levels
Answer : No throw, throws and states unchanged, throws and states changes but ok, exception unsafe.

Why do we need to declare noexcept / const in class's member function?
Answer : It is a constraint that
- const member function can only call other const member function, and
- noexcept member function can only call other noexcept member function, and
- const object can invoke only const member functions, while
- noexcept member function can be placed outside try-and-catch block

When we need to modify data member through a const member function, how? Can you think of an example.
Answer : Mutable. Mutex.

There are many exceptions in standard C++, how are they related? Why we need that?
Answer : They are related by inheritance. Therefore we can catch a hierarchy of exception using polymorphism.

Write the try-and-catch block.
Answer :

```
try
{
     fct_that_throws();
}
catch(const std::exception& ex)
{
     ...
}
```

Why do you put a const and a reference in the catch statement?
Answer : I put const because probably I wont modify it. Reference is for speed and polymorphism.

What is the time of back-insertion for vector, list and map?
Answer : O(1) for vector without resize, O(N) for vector with resize, O(1) for list and O(logN) for map.

Why do we need vector? Can we replace it with list?
Answer : No, list is slow as (1) new/delete is involved when insert/erase, (2) not contiguous, more cache missing

Why do we need list? Can we replace it with vector?
Answer : No, list is faster when we need to do a lot of in-the-middle insert/erase.

What library do you use in boost?
Answer : shared pointer, lexical, signal, posix time, asio, thread/mutex

Name some design pattern you used

- Singleton (with lazy initialization and double checking locked pattern)
- factory
- RAII
- NVI
- signal and slot
- iterator pattern
- CRTP
- producer consumer model
- thread pool.
- disruptot

Name some examples of RAII
Answer : `std::fstream, std::thread, std::lock_guard<T>, boost::scoped_ptr<T>`

Can you write your own memory management class?
Answer : Yes, for dynamic allocation for `scoped_ptr`, we can have our own stack-liked memory management (like ASM).
Answer : For transferable ownership, like `unique_ptr` and `shared_ptr`, we cannot implement stack-liked manager.

Tell me something about OOP.
Answer : Object is like a state machine.