Futures and Promises

Paradigms for parallellization( Paradigms cho sự song song)

**Solutions for coordinating work across multiple cores are many.**

Solutions cho việc điều phối công việc trên nhiều core là rất nhiều.

**Some are highly programmer-friendly and enable development of software that works exactly if it were running on a single core.**

Một vài Solutions rất thân thiện với lập trình viên và cho phép phát triển phần mềm mà hoạt động chính xác nếu nó đang chạy trên single core.

**For example the classic Unix process model is designed to keep each process in total isolation and relies on kernel code to maintain a separate virtual memory space per process. Unfortunately this increases the overhead at the OS level.**

Ví dụ , model process classic Unix được thiết kế để giữ mỗi tiến trình trong tổng số các tiến trình là độc lập và dựa vào code kernel để duy trì một space memory virtual riêng biệt trên một process. Thật không may , điều đó sẽ tăng chi phí tại OS level.

## Software development challenges( Thách thức của Software development)

## Hardware has changed to the point where the assumptions originally made on small numbers of CPU cores are no longer valid.

## Hardware đã thay đổi đến mức mà mọi giả định ban đầu được thực hiện trên một số lượng nhỏ của cores CPU là không còn hiệu lực.

## Processes are extremely self-contained but have high overhead.

## Các process rất độc lập nhưng có chi phí cao.

## Threads impose additional coordination costs on both the programmer and the application infrastructure, and are notoriously difficult to debug.

## Thread gánh chịu chi phí điều phối bổ sung cho người lập trình và cơ sở hạ tầng ứng dụng, và nổi tiếng là khó khăn để debug.

## Pure event-driven programming can result in codebases that are difficult to test and extend.

## Lập trình hướng sự kiện thuần túy có thể dẫn đến các codebase mà khó để test và mở rộng.

## An ideal solution would have: (một giải pháp lí tưởng sẽ có☺)

## Straightforward design to be comprehensible for program design and development

## Một thiết kế đơn giản để dễ hiểu cho thiết kế và phát triển chương trình.

## Minimum overhead on modern hardware

## Giảm thiểu chi phí cho modern hardware

## Low debugging costs (giảm chi phí debug)

## Solution: Seastar futures and promises

**The solution is a model known as “futures and promises”.**

Một solution là một mô hình được biết đến như là futures and promises”.

**A future is a data structure that represents some yet-undetermined result.**

Một future là một data structure mà đại diện cho một số kết quả chưa được xác định.

**A promise is the provider of this result.**

Một promise là một người cung cấp cho kết quả đó.

**It is sometimes helpful to think of a promise/future pair as a first-in first-out queue with a maximum length of one item, which may be used only once.**

Việc suy nghĩ về cặp promise/future như là một queue “First-in first-out” với độ dài tối đa của một item, mà chỉ được sử dụng một lần đôi khi rất là hữu ích.

**The promise is the producing end of the queue, while the future is the consuming end.**

**Like FIFOs, futures and promises are used to decouple the data producer and the data consumer.**

Giống như FIFO, future và promise được sử dụng để tách rời nhà sản xuất data và nhà tiêu dùng data.

**Basic futures and promises are implemented in the C++ standard library, and in Boost.**

Future và promise được triển khai trong thư viện chuẩn C++ và trong Boost.

**However, the optimized Seastar implementations of futures and promises are different.**

Tuy nhiên, Triển khai của futrure và promise trong Seastar là khác biệt.

**While the standard implementation targets coarse-grained tasks that may block and take a long time to complete, Seastar futures and promises are used to manage fine-grained, non-blocking tasks.**

Trong khi triển khai tiêu chuẩn hướng đến nhiệm vụ thô, có thể block và mất nhiều thời gian để hoàn thành, Future và promise của Seatar được sử dụng để quản lí các nhiệm vụ bất đồng bộ.

**In order to meet this requirement efficiently:** (để đáp ứng yêu cầu này một cách hiệu quả)

* Seastar futures and promises require no locking.
* Seastar futures and promises do not allocate memory.
* Seastar futures support continuations.