Operating Systems mutations, how and why integrate the user in the digital era ?

29 septembre 2014

Résumé

0.1 Introduction

0.2 State of the Art

keywords :
— Synchronous vs Asynchronous
— Blocking vs Non-Blocking
— Pull / Push
— Poll / Interupt
— Promises
— Futures
— Delay
— Callback
— Channels
— Reactive programming
— Interuptions
— Function Pointer

The computing world is based on this duality : Synchronous / Asynchronous.

Synchronous computing allow easy understanding of the computation structure, possible estimation of computation performances *a priori*, at the detriment of overall performance. While Asynchronous computing globally improves performances, but reduce the possiblity to understand the structure, or to measure *a priori* these performances.

We can make a comparison between Operating Systems and Javascript. See all the documents I write already on why a browser/social network is an Operating System. The compparison goes further : Operating Systems made a swtich from synchronous computation to asynchronous computating for performance issues using interuptions. UI system made the exact same shift for other reasons, for example with X11 with function pointers to call to render a window. Javascript made the same shift with callback.

Many tried to combine the advantages of both, with systems to transform one into another : to provide easy understanding of computation structure, while overall good performances.

We are going to show this duality through different stage or areas of programming.

0.2.1 Interruptions

In the early stages of assembler programming, to retrieve the result of a communication outside the system, the system polled the channel, waiting for the reply. This poll method has since been replaced by a method using interruption. Instead of waiting for the result, the system puts an address in an interrupt vector table. The execution resume at this address when the result is ready, triggering the interruption.

0.2.2 Function pointer

One of the first graphical interface system developed is X11, the window manager. To render a window, the program accountable for this window give X11 a function pointer. Each time the window needs to be repaint, X11 execute this function.

0.2.3 Promises

Remote Procedure Call (RPC) makes possible to distribute a system over multiple machine. When a part of the system needs a result from another part, it make a remote call to retrieve this result, and wait until it is available. To avoid wasting precious computing time waiting, mechanisms like promises replaced the RPC.

What is the role of control flow in this ?

0.3 Related Works

0.4 Conclusion