CS597: CONCURRENCY AND ALGORITHMS

Background and Introduction

Francis Joseph Serina

DEFINITION

Literal

- Concurrency is about two or more separate activities happening at the same time

In computers

- A system performing multiple independent activities in parallel, rather than sequentially

SINGLE CORE PROCESSORS

Historically, computers in the past can give the *illusion* of concurrency via **task switching**. The computer executes a chunk of one process, then a chunk of another process, and so on.

- Happens faster than we humans can perceive, hence, the illusion.

HARDWARE CONCURRENCY

Computers with multi-processors, multi-core processors, or both; are capable of genuinely running more than one task in parallel.

THREADS

Tasks or subtasks in software

Many processors can execute multiple threads in a single core via task switching. Multi-core processors still perform task switching.

Hardware Thread count is the true measure of how many independent tasks the hardware can genuinely run concurrently.

ANALOGY – EMPLOYEES WORKING ON A PROJECT

Remote

- Each employee has their own copy of the manual (resources)
- Additional effort needed for collaboration (communication)

Local

- Employees share a common manual (resources)
- Employees can draw ideas on paper or a white board (communication)

MULTI-PROCESS VS MULTI-THREADING

Multi-process

- Divide application into multiple, separate, single-threaded processes or application.
- Utilize standard IPC (inter-process communications) such as sockets, networking, files, etc.
- Can be delegated across different machines in a network

Multi-thread

- Single application with multiple threads
- Shared memory

PURPOSE OF CONCURRENCY

Separation of Concerns

Performance

PURPOSE - SEPARATION OF CONCERNS

Divide task based on purpose

- Video Rendering
- Audio Rendering
- UI
- Network I/O
- File I/O

PURPOSE - PERFORMANCE

Task Parallelism

- Reduce Total Running Time by dividing tasks and executing them in parallel

Data Parallelism

- Each thread performs the same operation on different portions of the data

EMBARRASSINGLY PARALLEL ALGORITHMS

A.K.A.

- Naturally Parallel
- Conveniently Concurrent

Highly scalable algorithms

- Increasing the hardware threads make the algorithms perform better

WHEN NOT TO USE CONCURRENCY

When the benefits do not outweigh the cost

Expected Performance Gain is insignificant

Harder to understand and debug

Increased complexity leads to more bugs

CONCURRENCY AND C++

Concurrency has been around for a long time

Different vendors have different implementations

Different compilers have different libraries

POSIX C Standard was popular

Boost added an OOP approach

C++11 standardized concurrency

SAMPLE – HELLO WORLD!

END