Computer Science - Architecture

Overview

How computers actually work?

Operations

Instruction Sets

ARM vs x86

Halting Problem

# Programs

Program

A program is a set of instructions intended to achive a goal. The instruction can be low level (complied by a machine into machine instructions), or high level (human readable, i.e. developer code).

Generally a program is self contained, being made of all the instructions needed to allow for independent execution. However, there are exceptions, such as when programs load and call other programs, or when they work in unison with other programs.

Processes

A process is the instance of a computer program which has been loaded from some store, into the computers memory and begins execution. The process description in memory contains the vital information such as:

* program counter - keeps track of current position in program (which instruction is currently being executed)
* registers
* variable stores
* file handles
* signals
* etc

A thread is the instruction executing part of a process, with common applications being single threaded, i.e. only executing one set of instructions at a time. Some applications are desgined to be multi-threaded, where mutliple sets of instructions are executed at the same time on different processors (or time-split single processor systems). All threads in within a process share the same process address space, and therefore can access the same memory description of the process (i.e. same variables, memory, etc).

Programs can also be designed to load and call other programs, which are then executed within the process of the original program in a linear single threaded way, or multi-threaded separate thread.

When several processes are required to work together it is called a system. However, processes interact only through system provided inter process communication mechanisms.

Threads

A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler (part of the OS). Multiple threads can exist within one process, running concurrently and sharing the same resources such as memory.

Each thread is an independent set of values for the processor registers, which includes the Instruction Pointer (IP), which controls what exectures in what order. A thread also include a stack pointer which points to a unique area of memory for each thread so, threads do not interfere with each other.

The value of the instruction pointer and the instruction stored at the location that it points to, defines the next value for the instruction pointer. For most instructions, this just advances the instruction pointer by the size of the instruction. However, control flow instructions can cause the instruction pointer to loop, skip, etc instructions. The path of executions can be seen as the instruction pointer weaving through the program code, hence the name 'thread'

Threads are often scheduled to execute in a order based on a prioritization scheme, although if there is only on thread per core, it is possible to not require a scheduler.

Thread Safe

Thread Safe is a concept applicable to multi-threaded programs, where a piece of code is designed so that it functions correctly during simultaneous execution by multiple threads. In particular if mutliple threads need to share the same piece of data, it can only be accessed by one thread at a time.

There are several ways to achieve thread safety:

* Re-entrancy - Code can be partially executed by one task, re-entered by another task, then resumed by the original
* Mutual exclusion - Access to shared data is serialized so that only one thread reads or writes a specific shared bit of data at one time. If a thread access mutliple pieces of shared data, it can lead to problems such as race conditions, deadlocks, livelocks, starvation, etc
* Thread-local storage - Each thread has its own private copy of variables, variables reatain their values across subroutine and code boundaries.
* Atomic operations - Shared data is accessed through atomic operations which cannot be interrupted by other threads. Atomic operations require special machine languauge , and form the basis of many thread locking mechanisms.