Systems programming

Systems programming, or **system programming**, is the activity of programming[1] computer system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly (e.g. word processor), whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or both (e.g. operating systems, computational science applications, game engines, industrial automation, and software as a service applications).[1]

Systems programming requires a great degree of hardware awareness. Its goal is to achieve efficient use of available resources, either because the software itself is performance critical or because even small efficiency improvements directly transform into significant savings of time or money.

Overview

The following attributes characterize systems programming:

- The programmer can make assumptions about the hardware and other properties of the system that the program runs on, and will often exploit those properties, for example by using an algorithm that is known to be efficient when used with specific hardware.
- Usually a low-level programming language or programming language dialect is used so that:
  - Programs can operate in resource-constrained environments
  - Programs can be efficient with little runtime overhead, possibly having either a small runtime library or none at all
  - Programs may use direct and "raw" control over memory access and control flow
  - The programmer may write parts of the program directly in assembly language
- Often systems programs cannot be run in a debugger. Running the program in a simulated environment can sometimes be used to reduce this problem.

Systems programming is sufficiently different from application programming that programmers tend to specialize in one or the other.

In systems programming, often limited programming facilities are available. The use of automatic garbage collection is not common and debugging is sometimes hard to do. The runtime library, if available at all, is usually far less powerful, and does less error checking. Because of those limitations, monitoring and logging are often used; operating systems may have extremely elaborate logging subsystems.

Implementing certain parts in operating systems and networking requires systems programming, for example implementing paging (virtual memory) or a device driver for an operating system.

History

Originally systems programmers invariably wrote in assembly language. Experiments with hardware support in high level languages in the late 1960s led to such languages as PL/S, BLISS, BCPL, and extended ALGOL for Burroughs large systems. Forth also has applications as a systems language. In the

1970s, C became widespread, aided by the growth of Unix. More recently a subset of C++ called Embedded C++ has seen some use, for instance it is used in the I/O Kit drivers of macOS.[2]

Alternative Meaning

For historical reasons, some organizations use the term *systems programmer* to describe a job function which would be more accurately termed systems administrator. This is particularly true in organizations whose computer resources have historically been dominated by mainframes, although the term is even used to describe job functions which do not involve mainframes. This usage arose because administration of IBM mainframes often involved the writing of custom assembler code (IBM's Basic Assembly Language (BAL)), which integrated with the operating system such as OS/MVS, DOS/VSE or VM/CMS. Indeed, some IBM software products had substantial code contributions from customer programming staff. This type of programming is progressively less common, but the term *systems programmer* is still the de facto job title for staff directly administering IBM mainframes.

See also

- Ousterhout's dichotomy
- System programming language
- Scripting language
- Interrupt handler

References

1. "Panel: Systems Programming in 2014 and Beyond" (https://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2014/Panel-Systems-Programming-Languages-in-2014-and-Beyond). Microsoft. Retrieved 4 December 2015.
2. Apple Inc (14 August 2009). "I/O Kit Device Driver Design Guidelines" (https://developer.apple.com/library/mac/documentation/DeviceDrivers/Conceptual/WritingDeviceDriver/CPluPlusRuntime/CPlusPlusRuntime.html#//apple_ref/doc/uid/TP30000695-BAJIBFDE). *developer.apple.com*. Apple Inc. Retrieved 16 September 2014.

Further reading

- Systems Programming (https://catalog.loc.gov/vwebv/holdingsInfo?&bibId=4177732) by John J. Donovan