Processes

Early computers allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's resources. In contrast, contemporary computer systems allow multiple programs to be loaded into memory and executed concurrently. This evolution required firmer control and more compartmentalization of the various programs; and these needs resulted in the notion of a **process**, which is a program in execution. A process is the unit of work in a modern computing system.

The more complex the operating system is, the more it is expected to do on behalf of its users. Although its main concern is the execution of user programs, it also needs to take care of various system tasks that are best done in user space, rather than within the kernel. A system therefore consists of a collection of processes, some executing user code, others executing operating system code. Potentially, all these processes can execute concurrently, with the CPU (or CPUs) multiplexed among them. In this chapter, you will read about what processes are, how they are represented in an operating system, and how they work.

Bibliographical Notes

Process creation, management, and IPC in UNIX and Windows systems, respectively, are discussed in [Robbins and Robbins (2003)] and [Russinovich et al. (2017)]. [Love (2010)] covers support for processes in the Linux kernel, and [Hart (2005)] covers Windows systems programming in detail. Coverage of the multiprocess model used in Google's Chrome can be found at http://blog.chromium.org/2008/09/multi-process-architecture.html.

Message passing for multicore systems is discussed in [Holland and Seltzer (2011)]. [Baumann et al. (2009)] describe performance issues in shared-memory and message-passing systems. [Levin (2013)] describes message passing in the Mach system, particularly with respect to macOS and iOS.

The implementation of RPCs is discussed by [Birrell and Nelson (1984)]. [Staunstrup (1982)] discusses procedure calls versus message-passing communication. [Harold (2005)] provides coverage of socket programming in Java. Details on Android RPCs can be found at https://developer.android.com/guide/components/aidl.html. [Hart (2005)]

and [Robbins and Robbins (2003)] cover pipes in Windows and UNIX systems, respectively.

Bibliography

[Baumann et al. (2009)] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, P. Simon, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: a new OS architecture for scalable multicore systems" (2009), pages 29–44.

[Birrell and Nelson (1984)] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, Volume 2, Number 1 (1984), pages 39–59.

[Harold (2005)] E. R. Harold, *Java Network Programming*, Third Edition, O'Reilly & Associates (2005).

[Hart (2005)] J. M. Hart, *Windows System Programming*, Third Edition, Addison-Wesley (2005).

[Holland and Seltzer (2011)] D. Holland and M. Seltzer, "Multicore OSes: looking forward from 1991, er, 2011", *Proceedings of the 13th USENIX conference on Hot topics in operating systems* (2011), pages 33–33.

[Levin (2013)] J. Levin, *Mac OS X and iOS Internals to the Apple's Core*, Wiley (2013).

[Love (2010)] R. Love, *Linux Kernel Development*, Third Edition, Developer's Library (2010).

[Robbins and Robbins (2003)] K. Robbins and S. Robbins, *Unix Systems Programming: Communication, Concurrency and Threads*, Second Edition, Prentice Hall (2003).

[Russinovich et al. (2017)] M. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals - Part 1*, Seventh Edition, Microsoft Press (2017).

[Staunstrup (1982)] J. Staunstrup, "Message Passing Communication Versus Procedure Call Communication", *Software—Practice and Experience*, Volume 12, Number 3 (1982), pages 223–234.