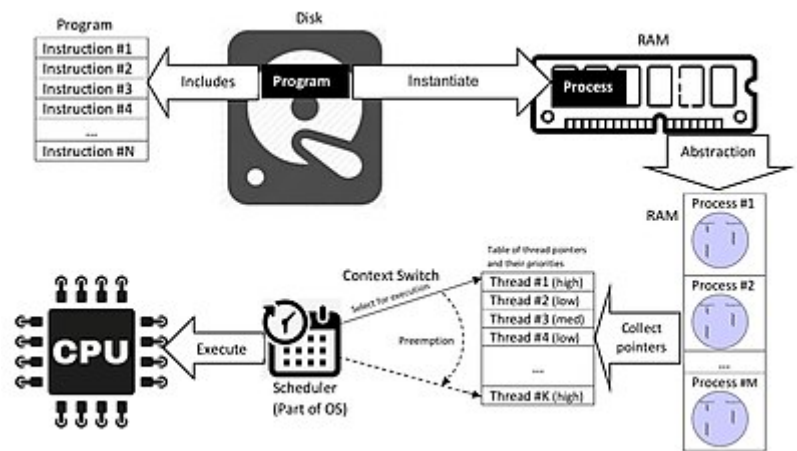


Process (computing)

In computing, a **process** is the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.^{[1][2]}

While a computer program is a passive collection of instructions typically stored in a file on disk, a process is the execution of those instructions after being loaded from the disk into memory. Several processes may be associated with the same program; for example, opening up several instances of the same program often results in more than one process being executed.



Program vs. Process vs. Thread
Scheduling, Preemption, Context Switching

Multitasking is a method to allow multiple processes to share processors (CPUs) and other system resources. Each CPU (core) executes a single task at a time. However, multitasking allows each processor to switch between tasks that are being executed without having to wait for each task to finish (preemption). Depending on the operating system implementation, switches could be performed when tasks initiate and wait for completion of input/output operations, when a task voluntarily yields the CPU, on hardware interrupts, and when the operating system scheduler decides that a process has expired its fair share of CPU time (e.g, by the Completely Fair Scheduler of the Linux kernel).

A common form of multitasking is provided by CPU's time-sharing that is a method for interleaving the execution of users' processes and threads, and even of independent kernel tasks - although the latter feature is feasible only in preemptive kernels such as Linux. Preemption has an important side effect for interactive processes that are given higher priority with respect to CPU bound processes, therefore users are immediately assigned computing resources at the simple pressing of a key or when moving a mouse. Furthermore, applications like video and music reproduction are given some kind of real-time priority, preempting any other lower priority process. In time-sharing systems, context switches are performed rapidly, which makes it seem like multiple processes are being executed simultaneously on the same processor. This simultaneous execution of multiple processes is called concurrency.

For security and reliability, most modern operating systems prevent direct communication between independent processes, providing strictly mediated and controlled inter-process communication functionality.

Contents

Representation

Multitasking and process management

Process states

External links

The operating system keeps its processes separate and allocates the resources they need, so that they are less likely to interfere with each other and cause system failures (e.g., deadlock or thrashing). The operating system may also provide mechanisms for inter-process communication to enable processes to interact in safe and

A process table as displayed by KDE System Guard

[1]

predictable ways.

Multitasking and process management

A multitasking operating system may just switch between processes to give the appearance of many processes executing simultaneously (that is, in parallel), though in fact only one process can be executing at any one time on a single CPU (unless the CPU has multiple cores, then multithreading or other similar technologies can be used).^[a]

It is usual to associate a single process with a main program, and child processes with any spin-off, parallel processes, which behave like asynchronous subroutines. A process is said to *own* resources, of which an *image* of its program (in memory) is one such resource. However, in multiprocessing systems *many* processes may run off of, or share, the same reentrant program at the same location in memory, but each process is said to own its own *image* of the program.

Processes are often called "tasks" in embedded operating systems. The sense of "process" (or task) is "something that takes up time", as opposed to "memory", which is "something that takes up space".^[b]

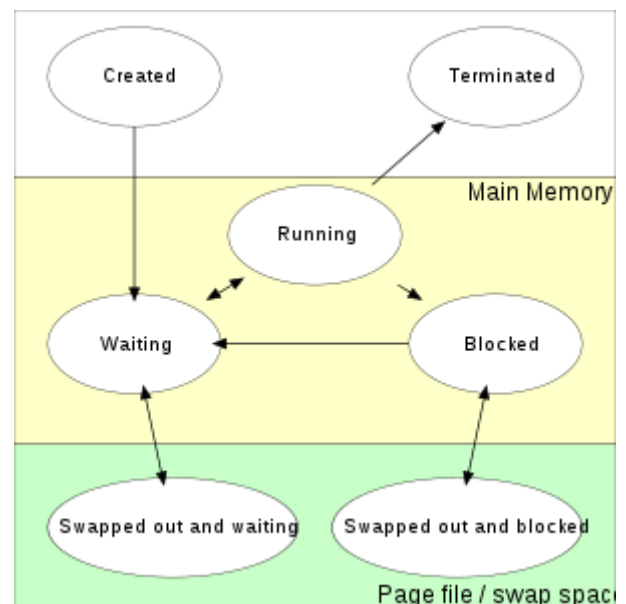
The above description applies to both processes managed by an operating system, and processes as defined by process calculi.

If a process requests something for which it must wait, it will be blocked. When the process is in the blocked state, it is eligible for swapping to disk, but this is transparent in a virtual memory system, where regions of a process's memory may be really on disk and not in main memory at any time. Note that even portions of active processes/tasks (executing programs) are eligible for swapping to disk, if the portions have not been used recently. Not all parts of an executing program and its data have to be in physical memory for the associated process to be active.

Process states

An operating system kernel that allows multitasking needs processes to have certain states. Names for these states are not standardised, but they have similar functionality.^[1]

- First, the process is "created" by being loaded from a secondary storage device (hard disk drive, CD-ROM, etc.) into main memory. After that the process scheduler assigns it the "waiting" state.
- While the process is "waiting", it waits for the scheduler to do a so-called context switch. The context switch loads the process into the processor and changes the state to "running" while the previously "running" process is stored in a "waiting" state.
- If a process in the "running" state needs to wait for a resource (wait for user input or file to open, for example), it is assigned the "blocked" state. The process state is changed back to "waiting" when the process no longer needs to wait (in a blocked state).



The various process states, displayed in a state diagram, with arrows indicating possible transitions between states.

- Once the process finishes execution, or is terminated by the operating system, it is no longer needed. The process is removed instantly or is moved to the "terminated" state. When removed, it just waits to be removed from main memory.^{[1][3]}

Inter-process communication

When processes need to communicate with each other they must share parts of their address spaces or use other forms of inter-process communication (IPC). For instance in a shell pipeline, the output of the first process need to pass to the second one, and so on; another example is a task that can be decomposed into cooperating but partially independent processes which can run at once (i.e., using concurrency, or true parallelism - the latter model is a particular case of concurrent execution and is feasible whenever enough CPU cores are available for all the processes that are ready to run).

It is even possible for two or more processes to be running on different machines that may run different operating system (OS), therefore some mechanisms for communication and synchronization (called communications protocols for distributed computing) are needed (e.g., the Message Passing Interface, often simply called MPI).

History

By the early 1960s, computer control software had evolved from monitor control software, for example IBSYS, to executive control software. Over time, computers got faster while computer time was still neither cheap nor fully utilized; such an environment made multiprogramming possible and necessary. Multiprogramming means that several programs run concurrently. At first, more than one program ran on a single processor, as a result of underlying uniprocessor computer architecture, and they shared scarce and limited hardware resources; consequently, the concurrency was of a *serial* nature. On later systems with multiple processors, multiple programs may run concurrently in *parallel*.

Programs consist of sequences of instructions for processors. A single processor can run only one instruction at a time: it is impossible to run more programs at the same time. A program might need some resource, such as an input device, which has a large delay, or a program might start some slow operation, such as sending output to a printer. This would lead to processor being "idle" (unused). To keep the processor busy at all times, the execution of such a program is halted and the operating system switches the processor to run another program. To the user, it will appear that the programs run at the same time (hence the term "parallel").

Shortly thereafter, the notion of a "program" was expanded to the notion of an "executing program and its context". The concept of a process was born, which also became necessary with the invention of re-entrant code. Threads came somewhat later. However, with the advent of concepts such as time-sharing, computer networks, and multiple-CPU shared memory computers, the old "multiprogramming" gave way to true multitasking, multiprocessing and, later, multithreading.

See also

- Child process
- Exit
- Fork
- Light-weight process
- Orphan process
- Parent process
- Process group

- Wait
- Working directory
- Zombie process

Notes

- a. Some modern CPUs combine two or more independent processors in a multi-core configuration and can execute several processes simultaneously. Another technique called simultaneous multithreading (used in Intel's Hyper-threading technology) can simulate simultaneous execution of multiple processes or threads.
- b. Tasks and processes refer essentially to the same entity. And, although they have somewhat different terminological histories, they have come to be used as synonyms. Today, the term process is generally preferred over task, except when referring to "multitasking", since the alternative term, "multiprocessing", is too easy to confuse with multiprocessor (which is a computer with two or more CPUs).


References

1. Silberschatz, Abraham; Cagne, Greg; Galvin, Peter Baer (2004). "Chapter 4. Processes". *Operating system concepts with Java* (Sixth ed.). John Wiley & Sons. ISBN 0-471-48905-0.
2. Vahalia, Uresh (1996). "Chapter 2. The Process and the Kernel". *UNIX Internals: The New Frontiers* (<https://archive.org/details/unixinternalsnew00vaha>). Prentice-Hall Inc. ISBN 0-13-101908-2.
3. Stallings, William (2005). *Operating Systems: internals and design principles* (5th ed.). Prentice Hall. ISBN 0-13-127837-1. (particularly chapter 3, section 3.2, "process states", including figure 3.9 "process state transition with suspend states")

Further reading

- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (2014). "Operating Systems: Three Easy Pieces (<https://web.archive.org/web/20200916133128/https://pages.cs.wisc.edu/~remzi/OSTEP/>)". Arpaci-Dusseau Books. Relevant chapters: Abstraction: The Process (<https://web.archive.org/web/20200916133128/https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf>) The Process API (<https://web.archive.org/web/20200916133128/https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf>)
- Gary D. Knott (1974) *A proposal for certain process management and intercommunication primitives* (<http://doi.acm.org/10.1145/775280.775282>) ACM SIGOPS Operating Systems Review. Volume 8, Issue 4 (October 1974). pp. 7 – 44

External links

-  Media related to Process (computing) at Wikimedia Commons
- Online Resources For Process Information (<http://www.processlibrary.com/>)
- Computer Process Information Database and Forum (<http://www.file.net/>)
- Process Models with Process Creation & Termination Methods (<https://osnote.space/process-models-with-process-creation-termination-methods.html>)

This page was last edited on 28 February 2021, at 13:02 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.