Architecture et Systèmes

Stefan Schwoon

Cours L3, 2015/16, ENS Cachan

Threads

Besides processes, threads are another way to implement concurrency.

Recall: Processes are units of execution with certain attributes and resources (id, memory, signals, . . .) that belong to the process itself. From the point of view of the system, processes function independently from one another.

Threads are units of execution *within* one process. They share certain resources (see next slides).

Threads are also called *lightweight processes*.

Private and shared resources

A thread owns its own

- program counter and call stack
- local (non-static) variables
- return value

All threads (of a process) share, e.g.,

- global (static) memory, heap
- open files (will discuss later)

Signals are “partially” shared (see later).

Working with threads

Standard library: Posix threads (Pthreads), defined in 1995

To compile programs with threads, include `pthreads.h` and use `gcc -pthread`.

See manpage `pthreads(7)` for overview

General notes about threads

A process can be thought of as a collection of threads
(by default, only one).

No hierarchy between threads inside a process.

Threads are “units of execution” scheduled by the kernel.

Each thread has a numerical identifier (type `pthread_t`).
This identifier can only be used within pthread functions.

In some implementations, threads are also associated with a PID.

Creation of a thread

Any thread can create another thread, using the function `pthread_create`.

Unlike `fork`, the current thread is not duplicated, instead `pthread_create` works almost like a procedure call (specify a procedure and one argument).

The starting procedure of a thread must be of type `void*` and accept one argument of type `void*`.

Termination of a thread

The thread lives until it terminates the procedure it started in, or calls `phread_exit`.

Similarly to processes, a thread may return an exit value.

Any thread can wait for termination of another thread (and query its value) using `pthread_join`.

A thread can be killed by another thread using `pthread_cancel`. Even the main thread can be terminated in this way!

Diverse notes

`pstree` shows threads in curly braces.

`pthread_detach` will make a thread non-joinable (it will not enter ‘zombie’ state and hence cannot return a value).

Lifetime of process and threads

The process lives as long as at least one thread is alive.

Also, any of the following terminates the process along with all its threads:

- a call to `exit` (by any thread);
- the main thread terminates the main procedure (with implicitly calls `exit`);
- the process or any thread is terminated by a signal.

Threads and Signals

The relation between threads and signals has evolved significantly in various implementations of Unix-like systems. The following is the POSIX specification (not respected by all implementations):

The disposition of signals (i.e., Term, Ign, etc) is *per process*, i.e. it is the same for all threads.

Hence, a terminating signal (e.g. sent by the kernel in response to certain actions of a thread, like `SIGFPE`, `SIGSEV`), will terminate the entire process unless caught.

The blocking mask is *per thread*, i.e. each thread has its own.

Signals can be sent to a process (*process-directed signal*). Unless the signal is ignored, it will be delivered to a thread that does not have it blocked. If no such thread exists currently, the signal will remain pending for that process.

Signals can be sent to a thread (*thread-directed signal*). In this case a signal can only be delivered to that thread and, if currently blocked, it remains pending for that thread.

Thread-directed signals are sent with `pthread_kill`.

Pros and cons

Advantages of threads over processes:

- Easier communication between threads (via variables, not files or signals).
- Thread creation/switching less costly
- Better use of system resources, e.g., two threads can operate on two cores.
- In short, more efficient.

Disadvantages:

- Less secure - a crash in one thread can terminate the entire program.
- Race conditions (conflicting write access).
- Certain system functions are not thread-safe (see *pthreads(7)* for a list).