Threading issues

Module 4
self study material

Operating systems 2020

1DT003, 1DT044 and 1DT096

karl.marklund@it.uu.se

Threading issues

- ★ Semantics of the `fork()` and `exec()` system calls
- ★ Thread cancellation of target thread
  - ◉ Asynchronous
  - ◉ Deferred
- ★ Signal handling
- ★ Thread-specific data

Semantics of `fork()` when using threads

Does `fork()` duplicate only the calling thread or all threads?

- ★ A new process shall be created with a single thread.
- ★ If a multi-threaded process calls `fork()`, the new process shall contain a replica of the calling thread and its entire address space.

Source: http://stackoverflow.com/a/18049303/3834375 2016-02-06

Semantics of exec() when using threads

If a process has multiple threads and one thread calls **`exec()`**, what happens?

- ★ If a thread invokes **`exec()`**, the executable (program) specified in the parameter to **`exec()`** will replace the entire process - including all threads.

Thread cancellation

Thread cancellation is the task of terminating a thread before it has completed.

Example 1

- Multiple threads concurrently **searching** through a **database** and one threads returns the result, the remaining threads might be canceled.

Example 2

- Often, a **web page** is loaded using several threads - each **image** is loaded in a separate thread etc. If a user presses the stop button in the browser, how can we cancel all the threads?

Thread cancellation issues

What can possible go wrong when a thread is cancelled? The difficulty with cancellation occurs in situations where:

- ★ resources have been allocated to a canceled thread
- ★ a thread is canceled while in the midst of updating data it is sharing with other threads.

Asynchronous cancellation

When a target thread is **cancelled** (terminated) **immediately** by another thread, this is called asynchronous cancellation. The cancellation is caused by something external to the target thread.

- ★ Often the OS will reclaim system resources from a canceled thread but will not reclaim all resources.
- ★ Cancelling a thread asynchronous may not free a necessary system-wide resource.

Deferred cancellation

Allows the target thread to periodically check if it should be cancelled.

- ★ The thread can perform the check for cancelation at a point at which it can be canceled safely.
- ★ The Pthreads threading API refers to such points as cancellation points.

Signals (1)

Signals are a limited form of inter-process communication used in Unix, Unix-like, and other POSIX-compliant operating systems

- ★ A signal is an **notification** sent to a process in order to notify it of an event that occurred.

- ★ When a signal is sent, the operating system **interrupts** the target process' **normal flow of execution** to deliver the signal.

- ★ If the process has previously registered a signal **handler**, that routine is executed. Otherwise, the default signal handler is executed.

Signals (2)

Signals can be synchronous or asynchronous.

Synchronous signals

- ★ Are delivered to the same process that performed the operation that caused the signal.
- ★ Examples of synchronous signals: **illegal memory** access and **division by zero**.

Asynchronous signals

- ★ Are generated by an event external to a running process.
- ★ Typically, an asynchronous signal is sent to another process.
- ★ Examples of asynchronous signals: **terminating** a process (ctrl-c), **timer expire**.

Threads and signal handling

Handling signals is more complicated in a multithreaded process - where should a signal be handled?

Options

- ★ Deliver the signal to the thread to which the signal applies.
- ★ Deliver the signal to every thread in the process.
- ★ Deliver the signal to certain threads in the process.
- ★ Assign a specific thread to receive all signals for the process.

Synchronous signals need to be delivered to the thread causing the signal.

Some asynchronous signals such as ctrl-c (terminate) should be sent to all threads.