Threads and the OS

Daniel Zappala

CS 360 Internet Programming
Brigham Young University

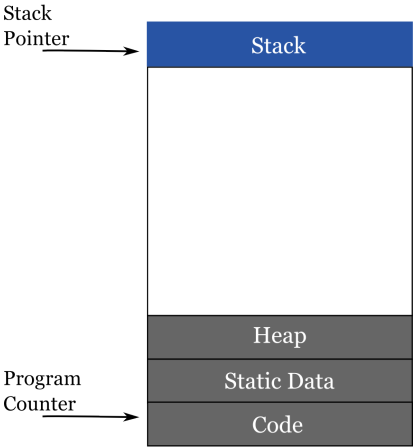
We all want to multitask

Multitasking

- processes: multiple apps
  - use web browser while also creating a presentation in Powerpoint
  - check Facebook while editing a video
- threads: multiple tasks inside the same app
  - browse through new photos while uploading others to Facebook
  - load a tab of a browser in background while reading contents of a different tab

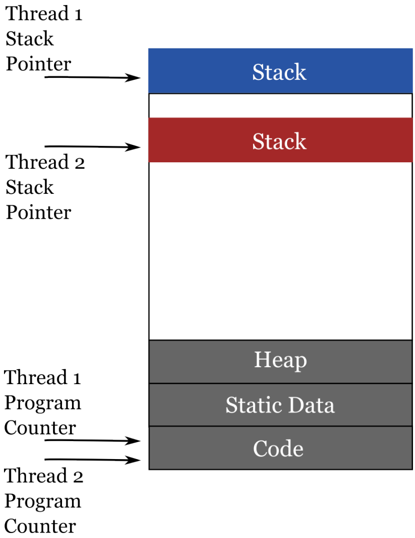
Process

- code
- data
- stack
- execution context
  - program counter
  - stack pointer
  - data registers



Thread

- belongs to a process
- shares code, data, stack with process
- has its own execution context
  - program counter
  - stack pointer
  - data registers



Benefits of Threads

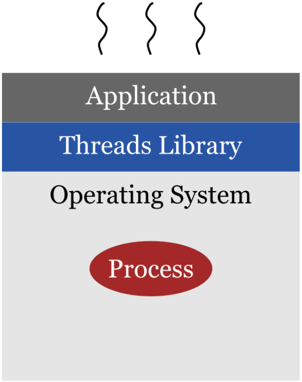
- faster to create a new thread than a process
- faster to switch between two threads within the same process
- more efficient communication between threads with shared memory
  - process communication requires protection and communication provided by kernel
  - threads can avoid the kernel
- parallel processing

Thread Support in Operating Systems

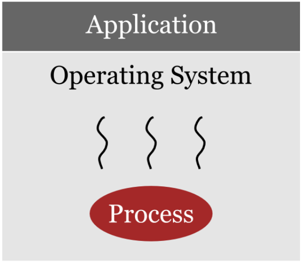
- MS-DOS: one process, one thread
- old Windows, UNIX: multiple user processes, but only one thread per process
- JVM: one process, multiple threads
- modern operating systems (Linux, Windows 2000+, Solaris, Mach): multiple threads per process

Types of Threads

User-Level and Kernel-Level Threads



User-Level Threads



Kernel-Level Threads

User-Level Threads

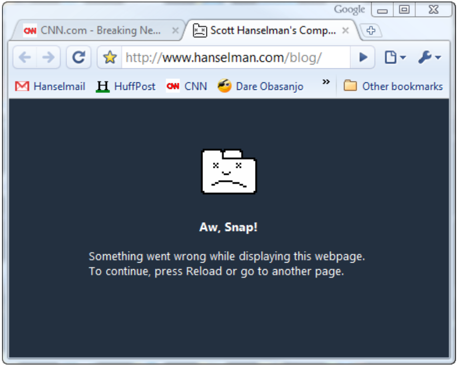
- all thread management done by the application
    - creating and destroying threads
    - thread communication
    - thread synchronization
    - thread scheduling
- runs in a single process, no kernel involvement
- advantages
    - efficient: no kernel mode switch to handle a different thread
    - application-specific scheduling
    - O/S independent
- disadvantages
    - thread system call blocks entire process
    - no multiprocessing: threads of the same process cannot run on different processors

Kernel-Level Threads

- thread management handled by kernel
- kernel schedules threads, not processes
- advantages
  - multiprocessing support
  - blocked thread doesn't block entire process
  - kernel can be multithreaded
- disadvantages
  - thread switching more expensive: requires mode switch

Why use Multiple Processes?

- separation of address space and resources
- one malfunctioning thread can halt the entire process
- Chrome often uses a new process for a new tab so that a crash in one tab is isolated from other tabs



C++11 Threads

Introduction

- allows you to create multiple threads of execution
- Linux
    - built on Pthreads, the POSIX threads library
    - 1:1 mapping to kernel level threads
    - compile application with *gcc/g++ -pthread -std=c++11*

Creating a Thread

- when a program starts, it runs in a single thread called the main thread
- create threads with `thread()`

```
template< class Function, class... Args >
explicit thread( Function&& f, Args&&... args );

thread t1(f,arg1,arg2);
```

- returns a thread object
- the new thread runs the given function with the given arguments immediately, and terminates by finishing this function

Joining a Thread

- waits for thread to terminate with `join()`

```
void join();

t1.join();
```

Getting a Thread ID

- get a thread ID with `get_id()`

```
thread::id get_id() const;

t1.get_id();
this_thread::get_id();
```

- returns the thread's thread identifier

Example Code

Example Code

- see example code for creating and joining threads

▸ GitHub