Threads and the OS

CS 360 Internet Programming

Daniel Zappala

CS 360 Internet Programming
Brigham Young University

We all want to multitask

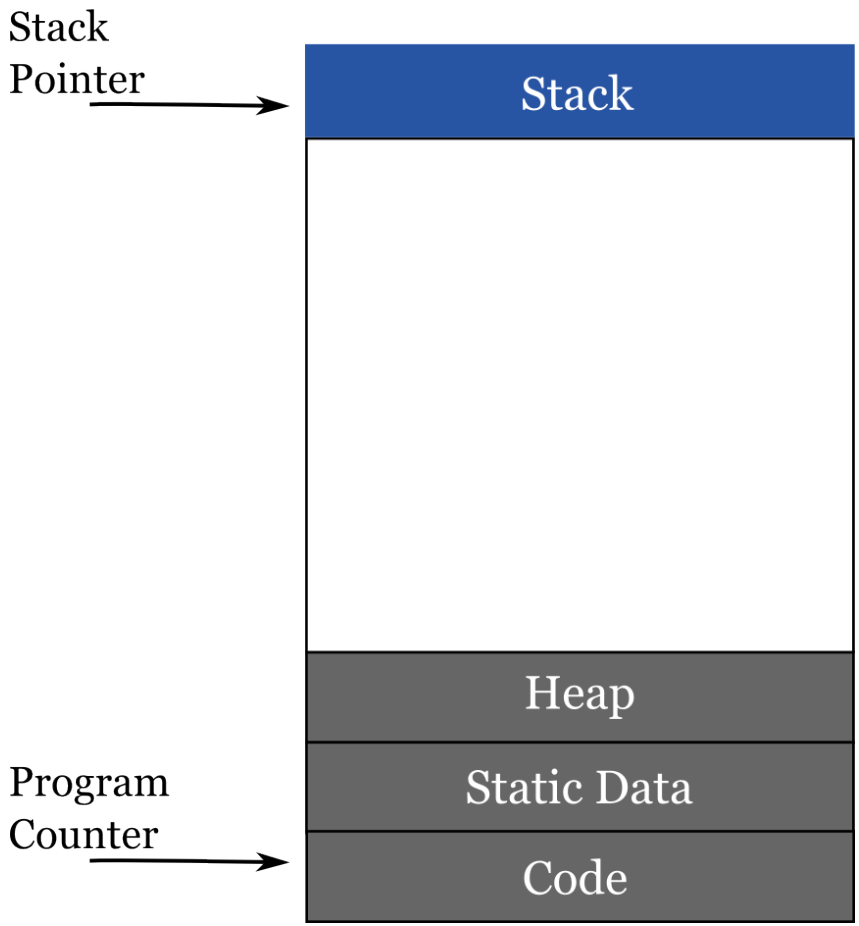
Multitasking

- 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

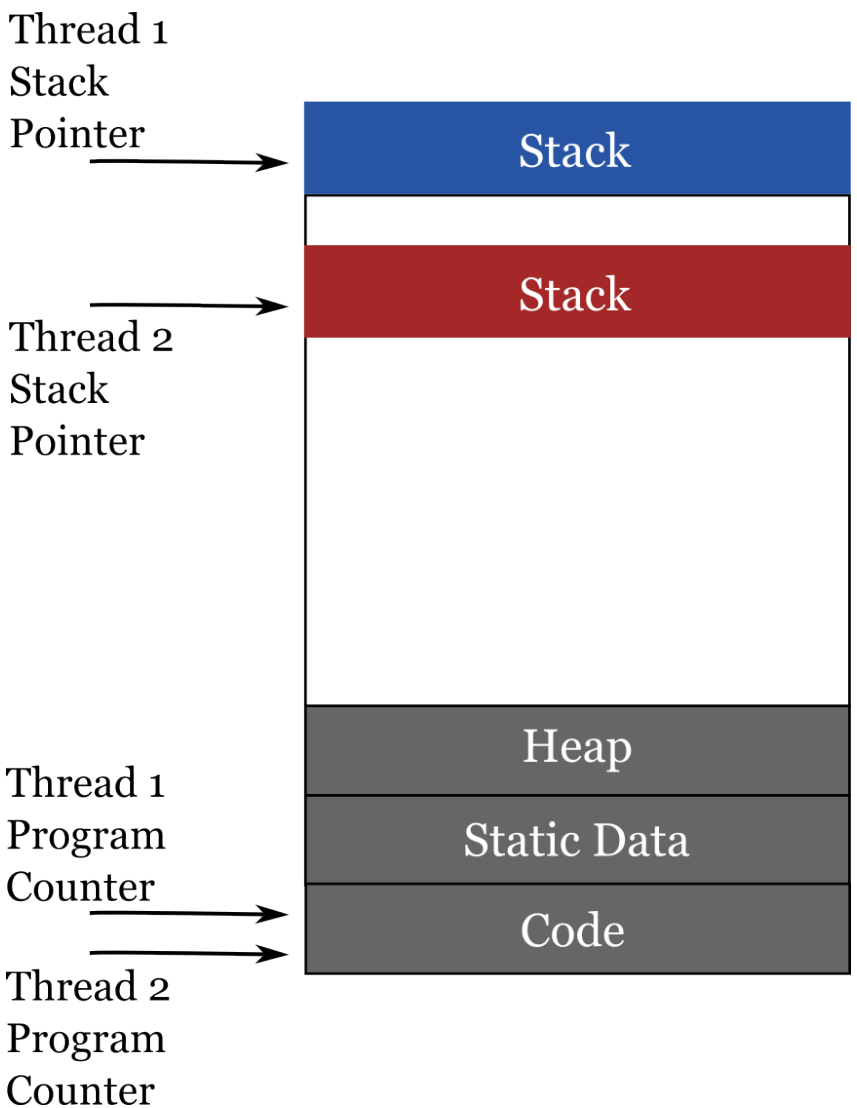
Process

- 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

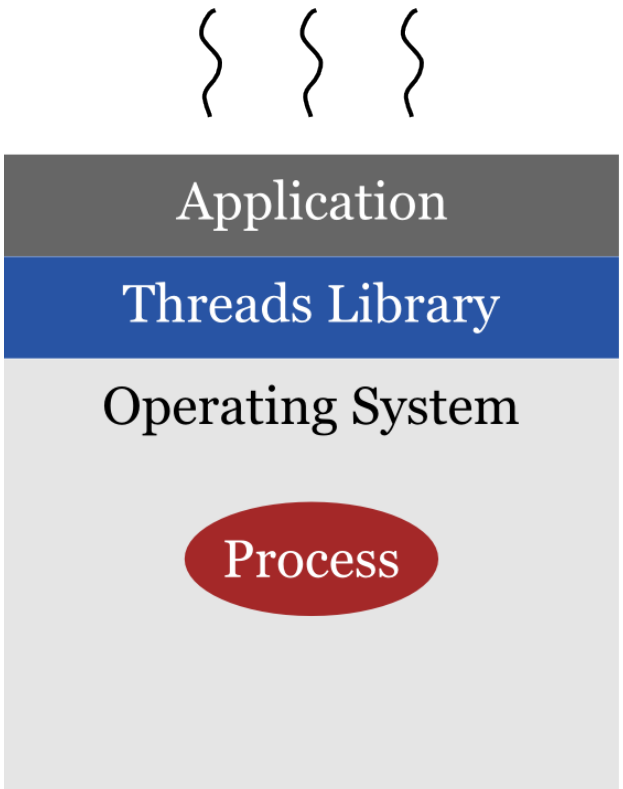
- faster to create a new thread than a process
- faster to switch between two threads within the same process
- more efficient communication between threads with shared memory
  - process communication requires protection and communication provided by kernel
  - threads can avoid the kernel
- parallel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

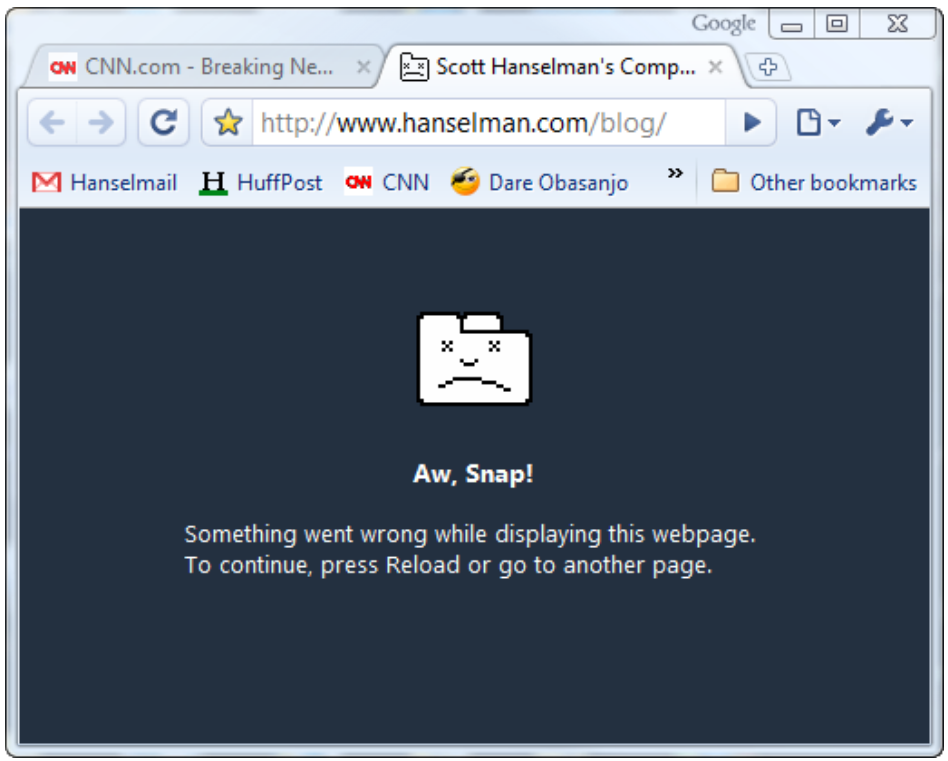
- all thread management done by the application
  - creating and destroying threads
  - thread communication
  - thread synchronization
  - thread scheduling
- runs in a single process, no kernel involvement
- advantages
  - efficient: no kernel mode switch to handle a different thread
  - application-specific 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



Pthreads

Introduction

- *Pthreads*: POSIX threads library
  - POSIX: IEEE standards defining OS API for Unix-like systems
- Linux
  - 1:1 mapping to kernel level threads
  - compile application with *gcc/g++ -pthread*
  - Native POSIX Thread Library (NPTL): heavily optimized, can run 100,000 threads simultaneously on a IA-32 which were started in two seconds; previously took 15 minutes
- http://www.llnl.gov/computing/tutorials/pthreads/

Creating a Thread

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

```
1 #include <pthread.h>
2
3 int pthread_create(pthread_t * thread, pthread_attr_t * attr,
4 void * (*start_routine)(void *), void * arg);
```

- the thread identifier is returned through the thread pointer
- the new thread runs the given start routine with the given arguments, terminates by finishing this routine
- attributes include priority, stack size, etc. - leave as default by passing a null pointer
- return value is normally zero, return positive error value otherwise

Joining a Thread

- wait for threads to terminate with `pthread_join()`

```
1  #include <pthread.h>
2
3  int pthread_join(pthread_t th, void **thread_return);
```

- specify thread identifier of thread to wait for
- return value of thread in given in returned pointer if non-null
- must call join to reclaim thread memory and thus avoid memory leaks

Getting a Thread ID

- get your own thread ID with `pthread_self()`

```
1 #include <pthread.h>
2
3 pthread_t pthread_self(void);
```

- returns the thread's thread identifier

Exiting a Thread

- exit a thread with `pthread_exit()`

```
1  #include <pthread.h>
2
3  void pthread_exit(void *retval);
```

- returned value can be any object that is not local to the thread

Example Code

Example Code

- see example code for creating and joining threads

▸ GitHub