Processes, Threads, and Concurrency

CSE 220: Systems Programming

Ethan Blanton & Carl Alphonce

Department of Computer Science and Engineering

University at Buffalo

Logical Control Flows

The text defines a logical control flow as:

[A] series of program counter values that [correspond] exclusively to instructions contained in [a program's] executable object file or in shared objects linked to [it] dynamically at run time.

The system provides each program with the illusion that its logical control flow runs on a dedicated computer.

Concurrency

Concurrency is when more than one logical control flow is present in the system at the same time.

Concurrent flows are logical control flows whose execution overlap in time.

Concurrent flows can be present even with only one processor.

Multiple flows can coexist on one processor via multitasking.

Multitasking time slices between multiple logical control flows.

- Each flow runs for a brief period of time, then is interrupted
- A context switch changes control to another flow
- The new flow runs for a brief period of time *(repeat)*

The Process

Our fundamental logical control flow abstraction is the process.

A process encapsulates:

- A set of instructions
- The memory they use
- The system resources they access
- …

All interactions with other processes are through the OS.

This is due to the dedicated computer model.

Threads

Threads provide a conceptually similar abstraction to processes.

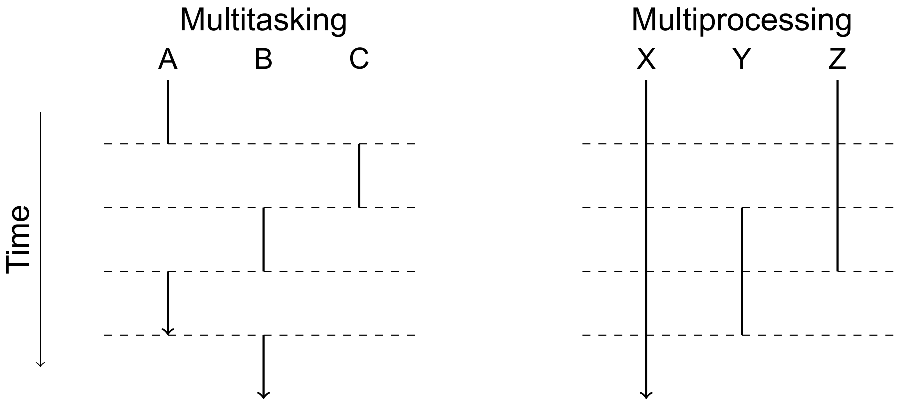
Threads also represent a logical control flow.

However:

- One process may have multiple threads
- Two threads within one process are much less isolated than two processes, or threads in different processes

In particular, threads within a process share a memory map.

Multitasking and Multiprocessing



Multitasking

Concurrent flows in a multitasking environment do not execute simultaneously.

However, from the point of view of any given flow, other flows are making progress while it executes.

Consider:

- Process A is executing at PC location L
- A context switch occurs, removing A from the CPU and switching to Process B
- Process B does something
- A context switch occurs, switching to Process A at location L

Process A will observe progress in Process B before and after L.

Multiprocessing

Concurrent flows in a multiprocessing environment may execute simultaneously.

Even with multiprocessing, multitasking may also be used.

This is typical for modern systems.

The operating system provides the illusion of a dedicated machine even to processes running simultaneously.

Concurrency and Separation

Concurrent flows may be related or unrelated in:

- Design
- Implementation
- Memory space
- Resource requirements
- Timing requirements
- …

When concurrent flows are completely unrelated, the dedicated computer abstraction provided by modern systems is both mostly complete and very appropriate.

When they are more related, it gets more complicated.

Motivation for Concurrency

There are many reasons to use concurrent flows:

- Making computational progress while blocked on a slow device
- Achieving rapid response to a particular condition (*e.g.*, human input, external event)
- Utilizing multiple physical processors
- …

In addition, simply taking advantage of the dedicated computer model to simplify design and implementation.

Processes

We have already seen process-level concurrency.

(Consider the chat client and server!)

Multiple processes may:

- Proceed independently on unrelated tasks
- Proceed independently on related tasks
- Cooperate on tasks

Independent, Unrelated Tasks

Independent, unrelated tasks are things like:

- Your windowing environment versus a terminal session
- A code editor and a music player

These tasks need not be aware of each other, and fit the dedicated computer model very nicely.

Independent, Related Tasks

Independent, related tasks might be:

- Make and the compiler
- Your chat client and the chat server
- A shell pipeline

These are programs that may or may not have been designed together, but are doing related work within the dedicated computer model.

Cooperating Tasks

Cooperating tasks could be:

- The individual tabs in a Chrome instance

These processes work closely together and may use the dedicated computer model for isolation, but are aware of each other.

Designing for Multiple Processes

A multi-process design can be robust and reliable.

The isolation in memory and resources provided by the system protects processes from certain faults in their neighbors.

Communication and cooperation can be expensive, though:

- Separate memory spaces protect, but also divide
- Many inter-process communication (IPC) mechanisms require interaction with the OS, which is slow

Most modern web browsers use multiple processes for this reason!

Threads

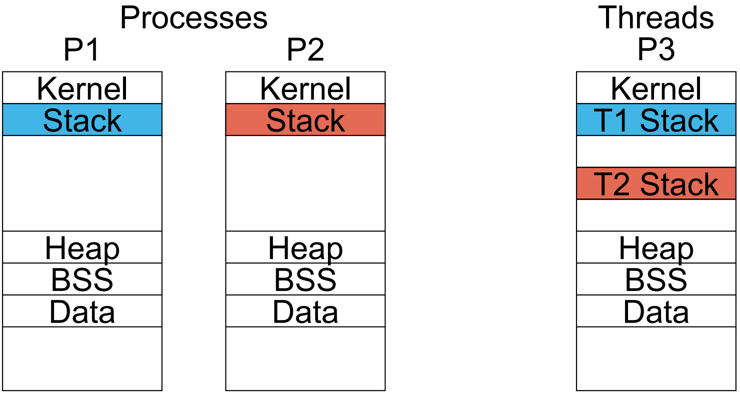
Thread are like processes that share almost everything.

They:

- Share memory
- Share system resources (such as open files)
- Run the same executable code
- …

Switching between threads is often less expensive than processes in a multitasking system.

Threads vs. Processes



Threading Advantages

Threads are much cheaper than processes:

- They share memory maps
- They share permissions and operating system resources
- Context switches between two threads in the same process are much less involved than between processes

Inter-thread communication is trivial, due to shared memory.

Threading Disadvantages

Concurrent access to shared resources is very tricky.

Many established APIs are not thread-safe.

(Over the next few lectures, think about a thread-safe `malloc()`![1])

Breaking down the dedicated computer model makes reasoning about process behavior harder.

[1]But you are *not* required to implement one!

Threading Use Cases

Threading is often appropriate for tasks which require:

- Very rapid change of control between parallel tasks
- Lots of large, shared data structures
- Blocking operations that do not inhibit other progress
- More rapid computation than can be performed on a single CPU

Multiple processes may also solve some of these problems.

The costs of threading must be weighed against its advantages on a case-by-case basis.

Summary

- Logical control flows are execution steps through programs.
- Concurrency is multiple logical control flows at one time.
- Multiprocessing versus Multitasking
- Processes versus Threads

Next Time …

- Races and Synchronization

References I

Required Readings

[1] Randal E. Bryant and David R. O'Hallaron. *Computer Science: A Programmer's Perspective*. Third Edition. Chapter 8: 8.2; Chapter 12: Intro, 12.1, 12.3. Pearson, 2016.

License

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.