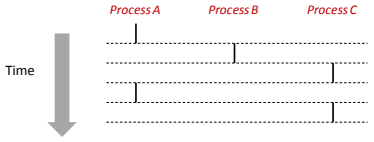
Lecture 25

Concurrent Programming

CPSC 275
Introduction to Computer Systems

Concurrent Processes

- Two processes *run concurrently* if their flows overlap in time
- Otherwise, they are *sequential*
- Examples (running on single core):



Concurrent Programming is Hard!

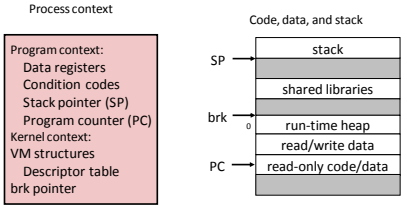
- The human mind tends to be sequential
- The notion of time is often misleading
- Thinking about all possible sequences of events in a computer system is error prone and frequently impossible

Concurrent Programming is Hard!

- Classical problem classes of concurrent programs:
  - ***Races:*** outcome depends on arbitrary scheduling decisions elsewhere in the system
  - ***Deadlock:*** improper resource allocation prevents forward progress
  - ***Starvation / Fairness***: external events and/or system scheduling decisions can prevent sub-task progress
- Many aspects of concurrent programming are beyond the scope of 275

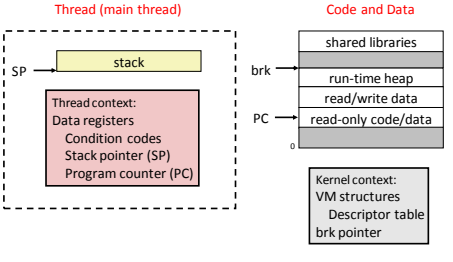
Traditional View of a Process

- Process = process context + code, data, and stack



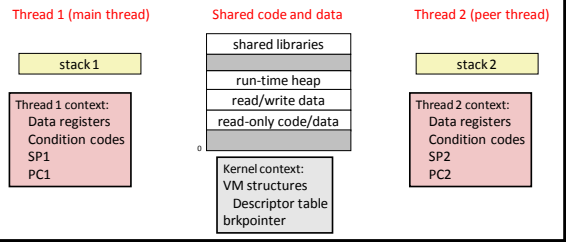
Alternate View of a Process

- Process = thread + code, data, and kernel context

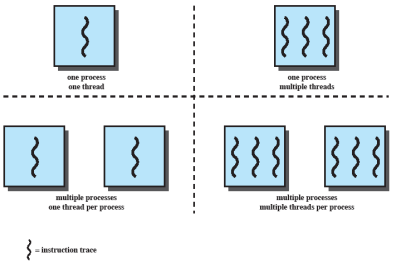


A Process With Multiple Threads

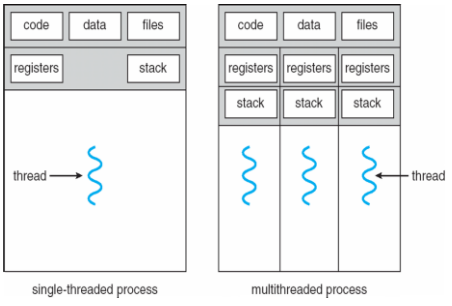
- Multiple threads can be associated with a process
  - Each thread has its own logical control flow
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own thread id (TID)



Processes and Threads

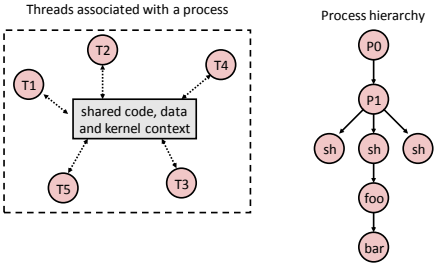


Threads



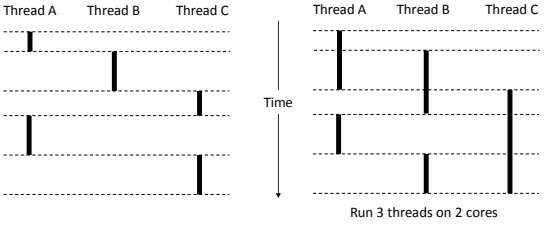
Logical View of Threads

- Threads associated with process form a pool of peers
  - Unlike processes which form a tree hierarchy



Thread Execution

- Single Core Processor
  - Simulate concurrency by time slicing
- Multi-Core Processor
  - Can have true concurrency



POSIX® Threads

- Also known as *Pthreads*.
- A standard for Unix-like operating systems.
- A library that can be linked with C programs.
- Specifies an application programming interface (API) for multi-threaded programming.

Posix Threads (Pthreads) Interface

- *Pthreads:* Standard interface for ~60 functions that manipulate threads from C programs
  - Creating and reaping threads
    - `pthread_create()`
    - `pthread_join()`
  - Determining your thread ID
    - `pthread_self()`
  - Terminating threads
    - `pthread_cancel()`
    - `pthread_exit()`
    - `exit()` [terminates all threads]
  - Synchronizing access to shared variables
    - `pthread_mutex_init`
    - `pthread_mutex_[un]lock`
    - `pthread_cond_init`
    - `pthread_cond_[timed]wait`

A Simple Example

Compile with:

$ gcc –o pth_hello pth_hello.c –lpthread

link in the Pthreads library

Run with:

$ ./pth_hello <#threads>

Starting the Threads

For each thread, call

```
int pthread_create(
        pthread_t *thread_p,        /* thread handler */
        const pthread_attr_t*  attr_p, /* thread attributes */
        void * (*start_routine) (void *),  /* thread function*/
        void *arg_p); /* arguments for thread function*/
```

A closer look (1)

```
int pthread_create (
        pthread_t *thread_p,
        const pthread_attr_t *attr_p,
        void *  (*start_routine ) (void *),
        void *  arg_p);
```

We won't be using, so we just pass NULL.

Allocate before calling.

A closer look (2)

```
int pthread_create (
        pthread_t*  thread_p,
        const pthread_attr_t *attr_p,
        void *  (*start_routine ) (void *),
        void *  arg_p) ;
```

Pointer to the argument that should be passed to the function *start_routine*.

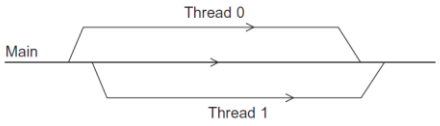
The function (task) that the thread is to run.

Function started by pthread_create

- Prototype:

  void *thread_function (void *args_p);
- void * can be cast to any pointer type in C.
- So args_p can point to a list containing one or more values needed by thread_function.

Running the Threads



Main thread forks and joins two threads.

Stopping the Threads

- We call the function pthread_join once for each thread.
- A single call to pthread_join will wait for the thread associated with the pthread_t object to complete.

Examples of Concurrent Programs

- Basic timing example
- Matrix multiplication