CISC 372: Parallel Computing
Threads

Stephen F. Siegel

Department of Computer and Information Sciences
University of Delaware

Operating System Concepts: Processes and Threads

- a thread is a "light-weight process"
- a process consists of ...
  - memory for the heap
  - resources the OS has allocated for the process: file descriptors, ...
  - environment variables
  - security information: what can this process access?
  - state information: is this process running, waiting, ...?
  - private address space (cannot be accessed directly by other processes)
    - In Unix, sockets can be used for inter-process communication
  - multiple threads
- a thread consists of ...
  - a stack (one frame for each function call)
  - program counter (reference to current position in the program)
- "thread" means "thread of control" in a program
- creating threads is "cheap" (compared to creating a whole process)
  - an OS can support many threads [see "Activity Monitor" in OS X]

Programming with threads in C

- originally, C did not support threads directly
- POSIX threads = Pthreads
  - an API for thread programming in C
  - POSIX is the "Unix" Standard (Portable Operating System Interface)
    - IEEE Std. 1003.1-2017, `https://pubs.opengroup.org/onlinepubs/9699919799/`
    - 4 volumes, thousands of pages
    - examples: Linux, BSD Unix (mostly conformant); macOS, AIX, HP-UX (certified)
  - by far the most common way to program threads in C
- C11 = C Standard from 2011 (C18 is current version)
  - added support for threads
  - some new syntax, mostly a standard library
  - similar in many respects to Pthreads
  - should be portable to any conforming C implementation supporting threads (not just Unix)
  - very complicated "memory model" (in comparison with Pthreads)
  - compilers, text books, tutorial, ...still catching up
  - some criticism from the user community on technical aspects

Other programming languages

- C++11, C++14, ...: threads
  - the C11 thread model was basically ported from C++11
- Java
  - always had threads built into the language
    - `Thread`, `synchronize`, wait(), notify, ...
  - still not easy to use
- Ada
  - 1983, 1995, 2005, 2012
  - always had concurrency: tasks
  - very elegant concurrency model
- many functional languages
  - Racket
  - Haskell
  - ...

First, more C: Function types

- if $T(x)$ declares x to have type T
  - then $T($`f()`$)$ declares `f` to have type *function-returning-T*
- $T($`f(t1, t2, ..., tn)`$)$
  - where the t_i are type names
  - further declares f to consume inputs of the named types
  - parameter names may also be included in the declaration but have no meaning
- $T($`f(void)`$)$ indicates that f has 0 parameters
- `()` binds tighter than `*`
  - `int *f();` : function-returning-pointer-to-int
  - `int (*f)();` : pointer-to-function-returning-int

Automatic conversion: functions

- an expression or parameter of type function-returning-T
  - is automatically converted to type pointer-to-function-returning-T
  - exceptions
    - operand of `sizeof`
    - operand of `&`
    - operand nof `_Alignof`
- this is a *function pointer*
  - in particular, the function used in a function call is really a function pointer
  - if you pass a function as an argument in a call
    - it is really a pointer to the function that you pass
- example
  - `int f(int);`
  - whenever `f` is used as an expression it is interpreted as a pointer to the function named `f`
  - `f(5)` : the `f` in this call is a pointer to the function

Exercise

Write a function `addmeup` which consumes

- a function `f` which consumes an `int` and returns an `int`

and returns

- $\sum_{i=0}^{9} \texttt{f}(i)$

Pthreads

- program begins executing with a single thread of control — just like a sequential program
- new threads are created dynamically by calling `pthread_create`
- you specify the function to run in the new thread with a function pointer
- global variables are shared
  - all threads can read and write to them
  - this is how threads communicate (as opposed to message-passing)
  - this must be done very carefully!
- local variables are private
- various synchronization mechanisms are provided by Pthreads
  - wait until another thread terminates
  - block until a lock becomes available. . . release the lock
  - wait until some condition becomes *true*
- requires a compiler and OS supporing Pthreads
  - all standard Unix systems

pthread_t

- a type
- a “thread ID” or reference to a running thread
- an opaque object
  - you cannot perform any operations on something of type `pthread_t`
  - you can only use that thing as an argument to functions in the Pthreads library
  - you don’t know how it is actually defined
    - it may be a pointer, or an int, or ...
    - completely up to the Pthread implementation

pthread_create

```
int pthread_create(pthread_t * thread,
                   pthread_attr_t * attr,
                   void *(*start_routine)(void*),
                   void * arg);
```

`pthread_create(thread, attr, start_routine, arg)`

`thread`	where ID of new thread is returned (`pthread_t*`)
`attr`	thread attributes, can be `NULL` (`pthread_attr_t*`)
`start_routine`	function to run in new thread (`void *(*)(void*)`)
`arg`	argument to provide to `start_routine` (`void*`)

Note: `start_routine` consumes a `void*` and returns a `void*`

pthread_join

```
int pthread_join(pthread_t thread, void **value_ptr);
```

```
pthread_join(thread, value_ptr)
```

`thread`	ID of thread to wait on (`pthread_t`)
`value_ptr`	address in which to store return val (`void**`)

- blocks until the specified thread terminates
- deallocates the thread resources
- similar to `MPI_Wait`

Hello, world!

See `hello/hello_pthread.c`.

Notes:

- the start function must consume and return a `void*`
- any pointer type can be converted to a `void*` and back again
- result is guaranteed to be equal to original pointer (C Standard)
- in this example: an `int*` is used for each thread
- need a separate one for each thread so each can have their own ID
- some people will convert a long to a `void*` and back
  - usually works
  - but result is not guaranteed to be the original `long`
  - not safe or portable!