Introduction to parallel computing

Shared Memory Programming with Pthreads (1)

Zhiao Shi (modifications by Will French)
Advanced Computing Center for Research & Education
Vanderbilt University
A Shared Memory System
Processes and Threads

• A process is an instance of a running program.
• Threads are analogous to a “light-weight” process and can access a shared memory space, in addition to their own local space.
• In a shared memory program a single process may have multiple threads of control.
• A process has one thread by default, if more are needed the programming must fork them from the “master” thread.
Thread-Safeness

• What makes an application thread-safe?
  • An application is thread-safe if it supports multiple threads of execution without shared memory space being “clobbered” by one or more threads.
  • The programmer is responsible for ensuring that shared memory is updated correctly so that multiple threads are not trying to access the same memory location simultaneously.
More About Threads

• Thread Scheduling

  • By default, the OS controls when a thread is scheduled to execute and which CPU core it will run on.
  
  • The order that threads execute portions of code relative to other threads often varies between instances of a program.
  
  • Programs should not depend on threads running in a particular order or on certain CPUs.
POSIX Threads

• POSIX: Portable Operating System Interface
• Also known as Pthreads.
• A library that can be linked with C programs.
• Specifies an application programming interface (API) for multi-threaded programming.
Pthreads:

• Pthread Library (60+ functions)
  • Thread management: create, exit, detach, join, . . .
  • Thread cancellation
  • Mutex locks: init, destroy, lock, unlock, . . .
  • Condition variables: init, destroy, wait, timed wait, . . .

• Programs must include the header file `pthread.h`

• Programs must be linked with the pthread library (`-lpthread`)
Fork-join model for executing threads in an application
What does the developer have to do?

• Decide how to decompose the computation into parallel parts.
• Create and destroy threads to support the decomposition.
• Add synchronization to make sure dependencies are covered.
The Pthreads API

• Three types of routines:
  • **Thread management**: create, terminate, join, and detach
  • **Mutexes**: mutual exclusion; creating, destroying, locking, and unlocking mutexes
  • **Condition variables**: event driven synchronization
  • Mutexes and condition variables relate to synchronization
# The Pthreads API naming convention

<table>
<thead>
<tr>
<th>Routine Prefix</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_</td>
<td>General pthread</td>
</tr>
<tr>
<td>pthread_attr_</td>
<td>Thread attributes</td>
</tr>
<tr>
<td>pthread_mutex_</td>
<td>mutex</td>
</tr>
<tr>
<td>pthread_mutexattr</td>
<td>Mutex attributes</td>
</tr>
<tr>
<td>pthread_cond_</td>
<td>Condition variables</td>
</tr>
<tr>
<td>pthread_condaddr</td>
<td>Conditional variable attributes</td>
</tr>
<tr>
<td>pthread_key_</td>
<td>Thread specific data keys</td>
</tr>
</tbody>
</table>
Starting the Threads

```c
int pthread_create ( 
    pthread_t* thread_p /* out */,
    const pthread_attr_t* attr_p /* in */,
    void* (*start_routine) ( void ) /* in */,
    void* arg_p /* in */) ;
```
**pthread_t objects**

- Opaque (its contents and data are hidden from the programmer).
- The actual data that they store is system-specific.
- Their data members aren’t directly accessible to user code.
- However, the Pthreads standard guarantees that a pthread_t object does store enough information to uniquely identify the thread with which it’s associated.
A closer look (1)

int pthread_create (  
    pthread_t* thread_p /* out */ ,  
    const pthread_attr_t* attr_p /* in */ ,  
    void* (*start_routine) ( void ) /* in */ ,  
    void* arg_p /* in */ ) ;

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

Allocate before calling.
A closer look (2)

int pthread_create ( 
    pthread_t*   thread_p  /* out */ , 
    const pthread_attr_t*  attr_p  /* in */ , 
    void*  (*start_routine ) ( void ) /* in */ , 
    void* arg_p /* in */ ) ;

Pointer to the argument that should be passed to the function \textit{start\_routine}.

The function that the thread is to run.

Returns 0 if OK, and other returns an error number.
pthread_create() example

....

void * myFunc( void * args ) {
    int * intArg = (int *) args;  // casting args as an (int *)
}

....

pthread_t myThread;
int myArg = 1;

int rc = pthread_create (&myThread, NULL, myFunc, (void *) &myArg);

....

Important note: pthread_create() is non-blocking, meaning the master thread will not wait for the thread it has created to complete myFunc().
Function started by pthread_create

• Prototype:
  ```c
  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.

• Similarly, the return value of thread_function can point to a list of one or more values.
Stopping the Threads

• The master thread calls the function `pthread_join` once for each thread and waits for the thread to complete its work.

• A single call to `pthread_join` will wait for the thread associated with the `pthread_t` object to complete.

```c
int pthread_join ( pthread_t thread_handle /*in*/,
                 void **status /*out*/);
```
Thread termination

void pthread_exit(void * status /*out*/)

• Terminates a thread.
• When used for master thread from main(), allows worker threads to complete before terminating process.
• When used for worker threads from thread function, can pass exit code back to master thread (combine with pthread_join()).
Thread Exiting

• A thread is terminated when it:
  • Returns normally from its starting routine.
  • Calls pthread_exit().
  • If the master thread finishes executing main() without calling pthread_exit(). This terminates the process and thus all threads.
  • The entire process is terminated due to an exit() call.
  • The thread is canceled by another thread with pthread_cancel().
Demo

• hello.c
Compiling a Pthread program

```
gcc -g -Wall -o hello hello.c -lpthread
```

link in the Pthreads library