OS Lab Tutorial

Thread

Today ...

- Pthreads
  - Create/Terminate
  - Obtain the returned value
- Synchronization
  - mutex
  - cond
- Some Programming Tips
- Assignment Spec

Threads VS. Processes

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2	1.2	0.6

Timings reflect 50,000 process/thread creations, units in seconds

Data Source: https://computing.llnl.gov/tutorials/pthreads/ Code: https://computing.llnl.gov/tutorials/pthreads/fork_vs_thread.txt

POSIX Threads (pthreads)

- POSIX - Portable Operating System Interface
  - maintain the compatibility between operating system
  - The standard, *POSIX.1c, Threads extensions (IEEEStd 1003.1 c-1995)*, defines an API for creating and manipulating threads.
- Standard Unix threading library
- Provides useful concurrent constructs
  - mutex
  - conditional variables
- Note
  - The POSIX semaphore API works with POSIX threads but is not part of threads standard. The semaphore procedures are prefixed by "sem_" instead of "pthread_" (http://en.wikipedia.org/wiki/POSIX_Threads)

pthread routines

- pthread_attr_init, pthread_attr_destroy
  - Initialize/destroy the attribute object of a thread
  - joinable/detached
- pthread_create
  - Create a thread
- pthread_join
  - wait for threads to finish
- pthread_exit
  - finish a thread
- pthread_cancel
  - Cancel a thread

Found no Man Page Info. ?

- Man Pages
  - The manual documents for POSIX pthread library may not be installed on some versions of the Linux system. You may have to use the online version or install it by yourself:
    - Linux Man Pages: http://linux.die.net/man/
      - Search by "Names" = pthread
    - Installation (Ubuntu):
      - $ sudo apt-get install manpages-posix-dev
      - $ sudo apt-get install glibc-doc
    - Then check the installation by:
      - $ man pthread_create

pthread_create

- <pthread.h>
- int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);
- Arguments
  - pointer to a thread ID, type: pthread_t
  - attribute object of a thread (could be NULL)
  - routine to be executed in a thread
  - arguments for the routine (could be NULL)

```
// Create a detached thread
pthread_attr_t attr; // thread attribute

// set thread detachstate attribute to DETACHED
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

// create the thread
pthread_create(&tid, &attr, start_routine, arg);
```

pthread_create

```
// From apue 2
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_sth( void *ptr ){
        char* message;
        message = (char*)ptr;
        printf("%s\n",message);
        return NULL;
}
```

```
int main()
{
    pthread_t thread1;
    char *message1 = "Thread 1";
    iret1 = pthread_create( &thread1, NULL, print_sth,
                                (void*) message1);
    pthread_join( thread1, NULL);
    printf("Thread 1 finished.\n”);
    return 0;
}
```

pthread_join

- Wait for a (joinable) thread to finish. If return value is not NULL, it copies the value into a location pointed to by *retval
- <pthread.h>
- int pthread_join(pthread_t thread, void **retval);
- Arguments
  - id of the created thread
  - returned value from the created thread

pthread_join

```
// From apue 2
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_sth( void *ptr ){
        char* message;
        message = (char*)ptr;
        printf("%s\n",message);
        return NULL;
}
```

```
int main()
{
    pthread_t thread1;
    char *message1 = "Thread 1";
    iret1 = pthread_create( &thread1, NULL, print_sth,
                                (void*) message1);
    // Blocked. Until thread1 finishes
    pthread_join( thread1, NULL);
    printf("Thread 1 finished.\n”);
    return 0;
}
```

pthread_exit and pthread_cancel

- <pthread.h>

- void pthread_exit(void *retval);
  - Terminate the calling thread and return the value

- int pthread_cancel(pthread_t thread)
  - Send a cancellation request to a thread

Programming Notification

- Include the corresponding header files
  - #include <pthread.h>
  - And others, ... e.g. "pthread_t" (id of a thread) is defined in <sys/types.h>
- Compile
  - $ gcc -Wall -o test pthread_test.c -lpthread
- Check the return value of a function call
  - Many functions write/record/set error information in a global variable errno when returning from errors
  - You can view the error message by examining errno

Errno, perror, strerror

- errno is an int variable declared in <errno.h>
- Each valid integer number (nonzero) represents an error code.
  - ($ man errno)

2	ENOENT	No such file or directory

- A good programming style is to ***always*** check the return value of a function and the associated errno.
- perror() / strerror() can be used to translate the number stored in errno into a human-readable string, i.e., *No such file or directory*.
  - strerror() is more useful when you want to customize your own output format.

```
err = pthread_create(&ntid, NULL, thr_fn, "new thread: ");
if (err != 0) {
        fprintf(stderr, "can't create thread: %s\n", strerror(err));
        exit(1);
}
```

Pthread Synchronization

- Semaphores (#include <semaphore.h>)
  - Resources counter
  - Allows a limited number of threads in the critical section
- Mutexes
  - Binary semaphores
  - Allows only one thread in the critical section
- Condition Variables
  - Communicate information about the status of some shared data

Mutex

- Header file: #include<pthread.h>
- pthread_mutex_init
- pthread_mutex_destroy
- pthread_mutex_lock
- pthread_mutex_unlock

pthread_mutex_init

- Set the attribute of a mutex
- int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
- We have three types of mutex
  - fast (default)
  - recursive
  - error check

pthread_mutex_lock

- If the mutex is unlocked: It becomes locked and owned by the calling thread
- If the mutex is locked by another thread, the calling thread is suspended until it is unlocked.
- What if the calling thread has already locked the mutex ?
  - pthread_mutex_lock(mutex);
  - pthread_mutex_lock(mutex);
  - ...

If the calling thread has already locked the mutex

- We have three types of mutex (statically defined below)
  - fast (default)
    - pthread_mutex_t fast = PTHREAD_MUTEX_INITIALIZER
    - Calling thread suspended on a locked mutex until it is unlocked (deadlock)
  - recursive
    - pthread_mutex_t recursive = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP
    - Calling thread succeeds and returns immediately if a mutex is locked, recording the number of times it has locked the mutex. (Used when you unlock the mutex)
  - error check
    - pthread_mutex_t errchk = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP
    - Calling thread returns immediately if a mutex is locked
- Note: They are different only when the mutex is already locked by the calling thread.

pthread_mutex_trylock

- It behaves identically to pthread_mutex_lock, except that it does not block the calling thread if the mutex is already locked by another thread(or by the calling thread in the case of a "fast" mutex)
- It returns immediately with error code "EBUSY".

pthread_mutex_unlock

- Unlocks the mutex. The mutex is assumed to be locked and owned by the calling thread
- Be careful if the mutex is "recursive" or "error check" type. They will have different behaviors.
  - Refer to MAN page:
    - http://linux.die.net/man/3/pthread_mutex_lock

Condition Variable

- Header file: #include<pthread.h>
- pthread_cond_init
- pthread_cond_wait
- pthread_cond_broadcast
- pthread_cond_signal

- Note: Condition Variable will always be used with a mutex

pthread_cond_init

- Unlike mutex, the attribute of a condition variable is actually ignored. So "NULL" is always specified.
- Or it can be initialized statically,
  - pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

pthread_cond_signal
pthread_cond_broadcast

- pthread_cond_signal
  - Restart one of the threads waiting on the condition
  - If there are more than one threads waiting on the same cond, it is not specified which thread will be restarted.

- pthread_cond_broadcast
  - Restart all the threads waiting on the condition variable

pthread_cond_wait

- Atomically unlocks the mutex (automatically)
- Waits for the cond to be signaled
- Thread is suspended and does not consume any CPU time until cond is signaled.
- Mutex must be locked by the calling thread before pthread_cond_wait()
- Before thread is recovered from the suspension at pthread_cond_wait(), it re-acquires (re-locks) mutex (automatically)

pthread_cond_destroy

- Test whether there are threads waiting on the condition variable

More helpful readings

- POSIX thread (pthread) libraries
- Synchronizing Threads with POSIX Semaphores
- POSIX Threads Programming
- Multi-Threaded Programming With POSIX Threads
- Multithreaded Programming (POSIX pthreads Tutorial)
- Pthread Tutorial