Network Programming TDC 561

Lecture # 6: Multithreaded Servers

Dr. Ehab S. Al-Shaer

School of Computer Science & Telecommunication
DePaul University
Chicago, IL

1

Introduction to Multithreaded Network Programming

- We are happy with fork(), Why threads?
  - fork() is expensive (30 slower in Solaris)
  - Multi-process is expensive (e.g.,context switch)
  - IPC mechanism is required between processes
  - Bounding to processors in MP machines
- Drawbacks
  - user-aware scheduling
  - different models
  - synchronization
  - Debugging

2

What is threads?

- Threads Concept
  - user space thread lib (-lthread or -lpthread)
  - kernel support
  - Share
    - process instructions
    - data
    - open files
    - signal handlers
    - current dir
    - user and grp ID
  - Do not share
    - thread id (tid)
    - priority
    - signal mask
    - errno
    - program counter and stack pointer)
    - stack (local var, and return address)

3

Applications Examples

- Inherently MT Programs
  - Independent tasks: GUI applications
  - Network or DB Servers: Netscape
  - Repetitive Tasks: concurrent simulations of similar entities
- Not obviously MT Programs
  - Numerical Programs (unless if they are devisable and MP machine is used): e.g., command line calculator

Threads and LWP

- LWP (lightweight process) is as a virtual CPU that is available for executing code
- LWP Features
  - run independent system call
  - can run in parallel in different CPUs
  - more LWP more CPU time
  - scheduling classes
  - kernel-level concurrency/scheduling
  - use thr_setconcurrency() to set up max LWPs.

Threads and LWP (cont.)

- Thread Features
  - completely under control of thread lib (kernel is not aware!)
  - completely in user space
  - number of magnitude faster than LWP
  - can be attached to LWPs
  - blocking system calls block all
  - user responsible for taking care of concurrency/scheduling
  - no kernel resource

Threads Kernel Scheduling

- Many threads on one LWP (HP and Dec)
  - + faster - blocking - no MP support
- One thread per LWP (NT and OS/2)
  - + no blocking + run simultaneously in MP
  - - slower (by kernel) - take some kernel resources
- Solaris Two-level Thread model:
  - Many-to-many: Many threads on many LWPs
  - one-to-one binding: One thread in one LWP

Comments from Experience ...

- Rule of thumb: you need as many LWPs as you have simultaneous blocking system calls. Use thr_setconcurrency().
- Avoid synchronization
- Signals (WATCH out)!!
- If no LWP is used, then watch out for thread starvation
- Solaris and POSIX threads compilation:

POSIX

```
#include <pthread.h>
 cc [ flag ... ] file ...  -lpthread [ libraries]
```

Solaris

```
#include <thread.h>
 cc [ flag ... ] file ...  -lthread [librararis ]
```


Threads Functions

POSIX

```
int pthread_create(pthread_t *new_thread_ID,
  const pthread_attr_t *attr,
  void * (*start_func)(void *), void *arg);
void pthread_exit(void *status);
```

Solaris

```
int thr_create(void *stack_base, size_t
  stack_size, void *(*start_func)(void *),
  void *arg, long flags, thread_t
  *new_thread_ID);
void thr_exit(void *status);
```

Threads Echo client and Server Examples

- Multithreaded Echo client
  - Stevens P.606
- Multithreaded Echo Server
  - Stevens: P. 607 and 610
- Comments:
  - passing arguments as void pointer
  - passing more than one argument
- Multithreaded Web Client
  - Stevens: P. 620

Threads Synchronization

- Problems with sharing variables
  - Race conditions
  - Deadlocks
- Solutions
  - mutual exclusion:
    - mutex
    - signals
    - semaphores
- Function and Examples

Threads Functions

POSIX:

```
int pthread_join(pthread_t target_thread, void
  **status);
 int pthread_detach(pthread_t threadID);
```

Solaris

```
int thr_join(thread_t target_thread, thread_t
  *departed, void **status);
```

POSIX

```
pthread_t pthread_self(void);
typedef unsigned int pthread_t;
```

Solaris

```
thread_t thr_self(void)
typedef unsigned int thread_t;
```

Pthreads Synchronization Functions

```
int pthread_mutex_init(pthread_mutex_t *mp,
  const pthread_mutexattr_t *attr);
  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t *mp);
int pthread_mutex_trylock(pthread_mutex_t *mp);
int pthread_mutex_unlock(pthread_mutex_t *mp);
int pthread_mutex_destroy(pthread_mutex_t *mp);

int pthread_cond_init(pthread_cond_t *cond,
  const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond,
  pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond,
  pthread_mutex_t *mutex,const struct timespec *abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);
```


Solaris Synchronization Functions

```
Solaris
#include <synch.h>
int mutex_init(mutex_t *mp, int type, void *arg);
int mutex_lock(mutex_t *mp);
int mutex_trylock(mutex_t *mp);
int mutex_unlock(mutex_t *mp);
int mutex_destroy(mutex_t *mp);

int cond_init(cond_t *cvp, int type, void *arg);
int cond_wait(cond_t *cvp, mutex_t *mp);
int cond_timedwait(cond_t *cvp, mutex_t *mp,
  timestruc_t *abstime);
int cond_signal(cond_t *cvp);
int cond_broadcast(cond_t *cvp);
int cond_destroy(cond_t *cvp);
```


Solaris Only

- Set concurrency as described before:

```
int thr_setconcurrency(int new_level);
int thr_getconcurrency(void);
```

- Others: `thr_suspend(), thr_continue()`
- See documentation in the home page or from ~ealshaer/tdc561/threads/docs in hawk, or from the course homepage

Threads Synchronization

- Problems with sharing variables
  - Race conditions
  - Deadlocks
- Solutions
  - mutual exclusion:
    - mutex
    - signals
    - semaphores
- Functions and Examples

Pthreads Synchronization Functions

```
int pthread_mutex_init(pthread_mutex_t *mp,
  const pthread_mutexattr_t *attr);
  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t *mp);
int pthread_mutex_trylock(pthread_mutex_t *mp);
int pthread_mutex_unlock(pthread_mutex_t *mp);
int pthread_mutex_destroy(pthread_mutex_t *mp);

int pthread_cond_init(pthread_cond_t *cond,
  const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond,
  pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond,
  pthread_mutex_t *mutex,const struct timespec *abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);
```


Solaris Synchronization Functions

```
Solaris
#include <synch.h>
int mutex_init(mutex_t *mp, int type, void *arg);
int mutex_lock(mutex_t *mp);
int mutex_trylock(mutex_t *mp);
int mutex_unlock(mutex_t *mp);
int mutex_destroy(mutex_t *mp);

int cond_init(cond_t *cvp, int type, void *arg);
int cond_wait(cond_t *cvp, mutex_t *mp);
int cond_timedwait(cond_t *cvp, mutex_t *mp,
  timestruc_t *abstime);
int cond_signal(cond_t *cvp);
int cond_broadcast(cond_t *cvp);
int cond_destroy(cond_t *cvp);
```