CISC 7310X
C04d: Thread Issues

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- This slides are a revision of the slides by the authors of the textbook

Outline

- ## Threading Issues

  - Semantics of fork() and exec()

  - Signal handling

  - Thread cancellation

  - Thread-local storage

  - Scheduler Activations

- ## Operating System Examples

Thread Issues

- Semantics of **fork()** and **exec()** system calls

- Signal handling

  - Synchronous and asynchronous

- Thread cancellation of target thread

  - Asynchronous or deferred

- Thread-local storage

- Scheduler Activations

Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?

  - Some UNIXes have two versions of fork

- **exec()** usually works as normal – replace the running process including all threads

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred.

- A signal handler is used to process signals

  - Signal is generated by particular event

  - Signal is delivered to a process

  - Signal is handled by one of two signal handlers:

    - default

    - user-defined

- Every signal has default handler that kernel runs when handling signal

  - User-defined signal handler can override default

  - For single-threaded, signal delivered to process

Signal Handling

- Where should a signal be delivered for multi-threaded?

  - Deliver the signal to the thread to which the signal applies

  - Deliver the signal to every thread in the process

  - Deliver the signal to certain threads in the process

  - Assign a specific thread to receive all signals for the process

Thread Cancellation

- Terminating a thread before it has finished

- Thread to be canceled is **target thread**

- Two general approaches:

  - **Asynchronous cancellation** terminates the target thread immediately

  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Thread Cancellation: Pthreads

• Pthread code to create and cancel a thread:

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

    . . .

/* cancel the thread */
pthread_cancel(tid);

/* wait for the thread to terminate */
pthread_join(tid,NULL);
```

Thread Cancellation: Pthreads

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it

- Default type is deferred
  - Cancellation only occurs when thread reaches **cancellation point**
    - i.e. `pthread_testcancel()`
    - Then **cleanup handler** is invoked

- On Linux systems, thread cancellation is handled through signals

Thread Cancellation: Java

- Deferred cancellation uses the `interrupt()` method, which sets the interrupted status of a thread.

```
Thread worker;

   . . .

/* set the interruption status of the thread */
worker.interrupt()
```

A thread can then check to see if it has been interrupted:

```
while (!Thread.currentThread().isInterrupted()) {
     . . .
}
```

Thread-local Storage

- **Thread-local storage** (**TLS**) allows each thread to have its own copy of data

- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

- Different from local variables

  - Local variables visible only during single function invocation

  - TLS visible across function invocations

- Similar to `static` data

  - TLS is unique to each thread

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application

- Typically use an intermediate data structure between user and kernel threads – **lightweight process** (**LWP**)

    - Appears to be a virtual processor on which process can schedule user thread to run

    - Each LWP attached to kernel thread

    - How many LWPs to create?

- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library

- This communication allows an application to maintain the correct number kernel threads

CUNY | Brooklyn College
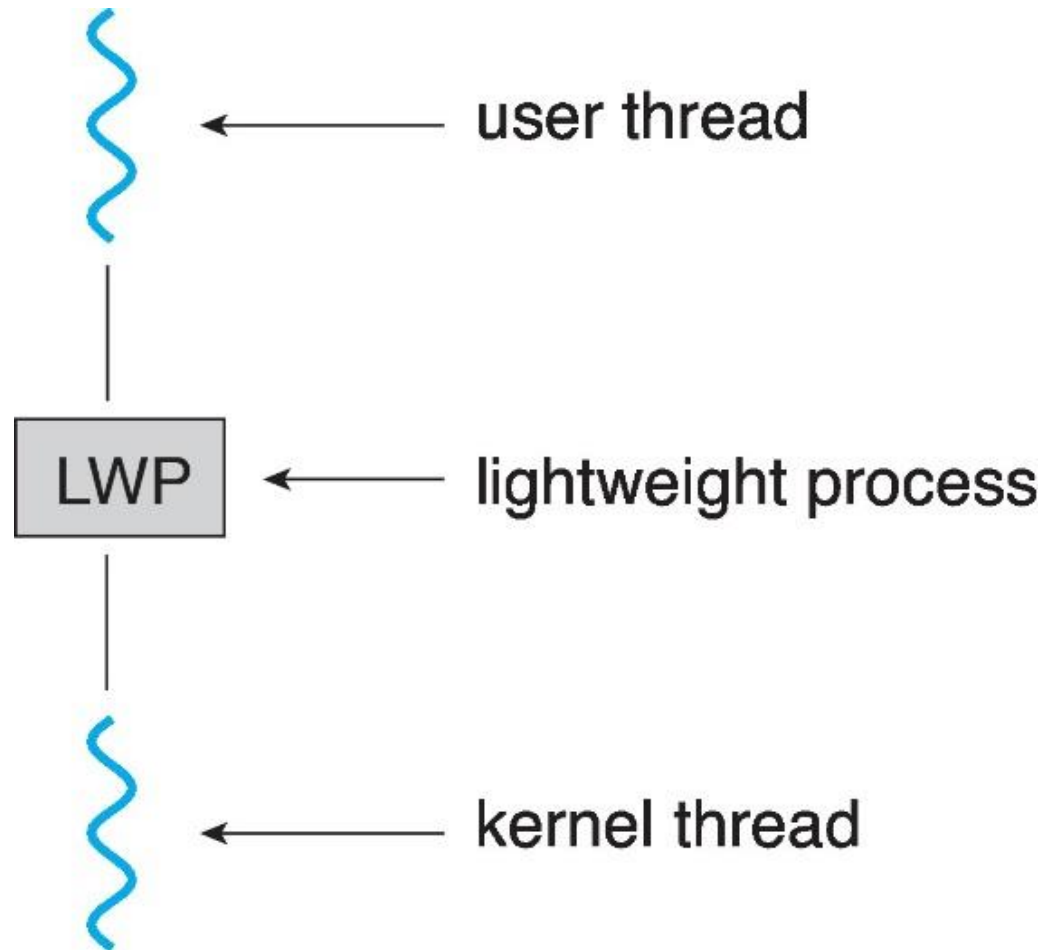
Questions?

- Threading Issues
  - Semantics of fork() and exec()
  - Signal handling
  - Thread cancellation
  - Thread-local storage
  - Scheduler Activations