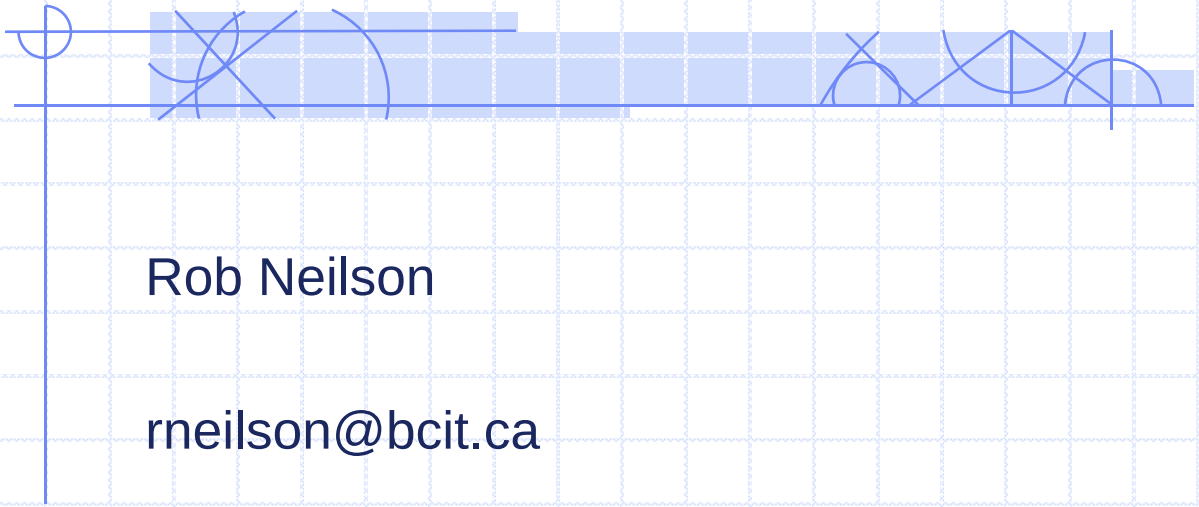
COMP 4735: Operating Systems

Lecture 7.1: Threads Part 2



Rob Neilson

rneilson@bcit.ca

Reading

- The following sections should be read before next Monday
  - it would also help to read this before your lab this week
  - Textbook Sections: 2.2 (Theory Part)
    10.3.3 (Case Study Part)
- Yes, there will be a quiz next Monday in lecture on the above sections
  - A sample quiz will be posted on webct on Friday
  - This quiz (Quiz 4) covers material from all of the sections listed above

Agenda

Key concepts for this lesson:

- User Space Threads
- Kernel Space Threads
- Threads in Linux

Question 5

5. Assume that you compile and run the program shown below on a multi-threaded OS. How many threads are created?

a) zero
b) one
c) two
d) three
e) four

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void *bye (void *id) {
   printf ("I am about to die ... %d\n", id);
   pthread_exit(0);
}

int main (int argc, char *argv[])
{
   int i=1;
   int rc;
   pthread_t threads[3];
   rc=pthread_create(&threads[i], 0, bye, (void *)i);
   exit(0);
}
```

Question 6

6. Which of the following Pthread library calls would you make if you want your program to block and wait for a specific thread to finish executing?

   a) **`Pthread_block()`**

   b) **`Pthread_exit()`**

   c) **`Pthread_join()`**

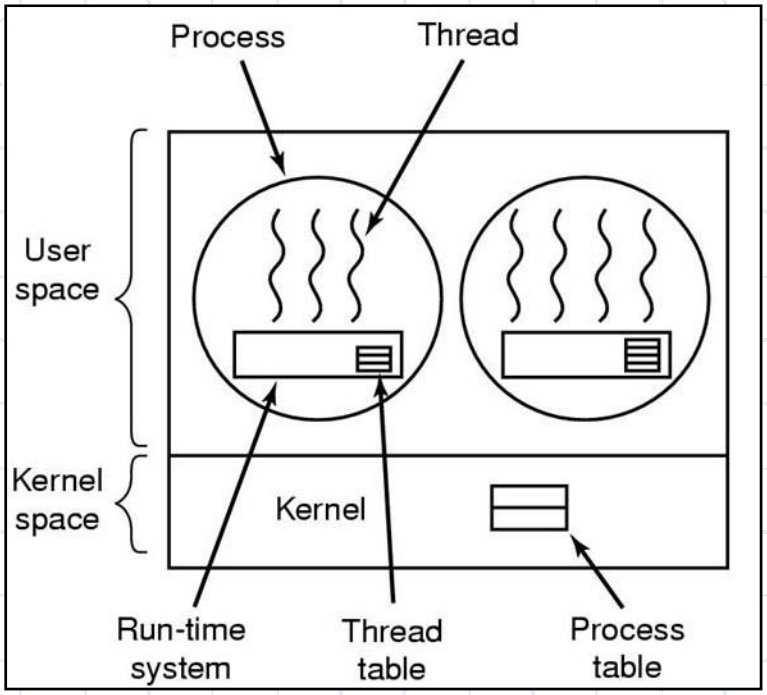
   d) **`Pthread_wait()`**

   e) **`Pthread_yield()`**

   f) none of the above

User Space Threads

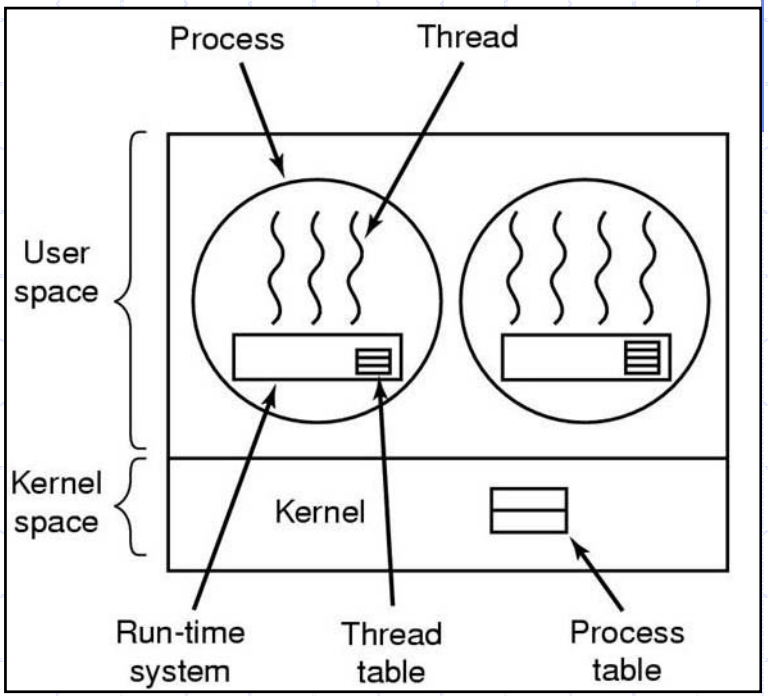
- threads are implemented in a package (library)
- a multi-threaded program must include the library; (eg: Pthreads)
- the code in the library implements everything that is needed to support multi-threading
  - thread table
  - thread scheduler
  - thread state machine
  - all of this information is managed by a thread ***run-time system***



- ***the threads all execute within a process***
- ***the CPU is completely unaware of the threads***
- ***the kernel schedules processes not threads***

User Space Threads - Management

- threads are managed by the run-time system
- when a thread wants to create another, a library call is made
- this library call is local and is handled within the process
- the thread tables are managed within the process - without ever trapping to the kernel



User Space Threads - Benefits

- thread switching is very fast as there is no need to trap to the kernel
- can run on legacy systems that do not support kernel threads
- scale better (ie: do not affect system performance as much as kernel threads when the number of threads gets large)
  - ... because they don't require kernel memory for tables etc

enable lightest-weight, fastest method of switching threads of execution

User Space Threads - Drawbacks

- blocking system calls cause *entire process to block*
  - this means that *all threads in a process are blocked* every time any thread wants to do IO or has a page fault
  - unfortunately the biggest benefit of multi-threading is for programs that do lots of IO ... so we have a *paradox*

- threads have to *voluntarily give up the CPU*
  - if a thread hogs the CPU, other threads in the process will not run
  - the thread scheduler has *no way of pre-empting a thread*
  - the clock in the kernel can pre-empt the entire process, but not a specific thread
  - the threads have to call *thread_yield()* to voluntarily give up the CPU

Question 7

7. Assume you have a process that is running a multi-threaded program on a system that uses user-space threads.

   Which one of the following events will cause all the threads in the process to block?

   a) the garbage collector thread starts executing

   b) a thread calls thread_create() to create a new thread

   c) a thread calls thread_exit to terminate itself

   d) a thread calls read() to load a record from a file

   e) a thread is scheduled for execution

   f) none of the above

Question 8

8. Assume that a new thread is created on a system that runs a user space threads package. The thread is put in the READY state. Where will this status information (READY) be stored?

a) in a process table in the process

b) in a process table in the thread

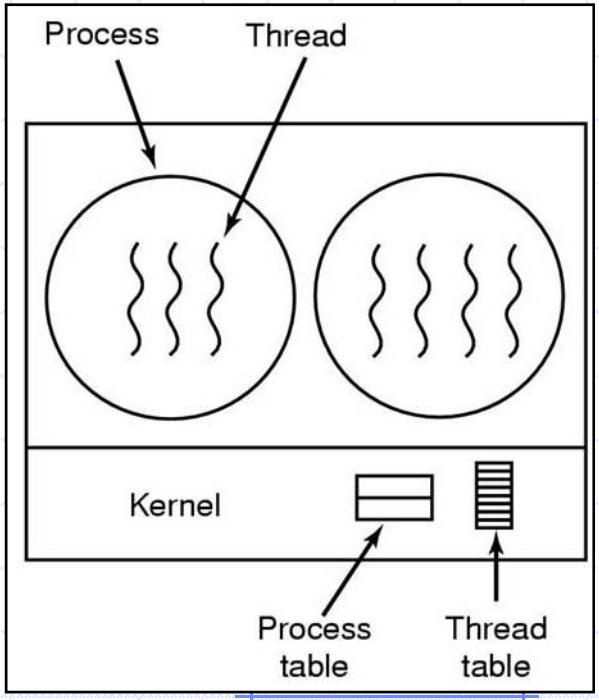
c) in a thread table in the kernel

d) in a thread table in the process

e) none of the above

Kernel-Space Threads

- user space threads were implemented in a library, and the OS knew nothing about them
- ***kernel threads are implemented inside the kernel itself***
- the OS designers have to create a process table extension (ie: *thread table*), and change the kernel scheduling and context switching components to operate on threads
- ***the kernel schedules threads - not processes***
- ***the CPU manages the threads, including:***
  - ***creation***
  - ***termination***
  - ***scheduling***
- ***a program must trap to the kernel if it needs to create or terminate a thread***



Kernel-Space Threads - Advantages

- *The disadvantages of User-space threads are the advantages of kernel-space threads*, for example:
- blocking system calls block individual threads, not processes
  - only the thread that wants to do IO is blocked, and *other threads in the process are free to run* and perform other activities
  - this allows *programs with significant IO to realize the largest increases in performance* through multi-threading (ie: there is the potential for increased CPU / IO overlap within processes)
- the kernel will schedule the threads individually
  - the scheduler is in the kernel, and can pre-empt threads
  - a single thread is not able to monopolize the CPU,
  - it is no longer necessary to call thread_yield()

enable maximum benefits of multi-core and multi-processor platforms

Kernel-Space Threads - Drawbacks

- thread management requires the program to *trap to the kernel*, which is *slower*
- thread support must be *built into the OS* which is costly and *technically challenging*
- there can be many, many, threads running concurrently on a modern system
  - requires a *lot of kernel memory*
  - kernel memory is expensive and (typically) pre-allocated
  - this increases the memory management burden within the kernel

Question 9

9. Which one of the following is an advantage of kernel space threads?

   a) superior performance for non-IO bound applications

   b) fast thread switching

   c) supports micro-kernel design concepts

   d) blocking system calls do not affect other threads in the same process

Question 10

10. A program with multiple threads of execution that has hard real-time requirements ( ) should ***not*** be developed on which of the following infrastructures:

a) User-space threads

b) Kernel-space threads

c) Classic process model (no threads)

d) it doesn't matter, any of the above approaches will work

What type of threads do different systems use?

- User-space threads
  - POSIX *Pthreads (used on most traditional Unix platforms)*
  - Solaris *threads*

- Kernel-space threads
  - Windows 95/98/NT/2000
  - Solaris
  - Linux
  - Mac OS-X 10.4 (Tiger)

Linux Threads

- Linux allows the programmer to manage the creation of threads, processes and/or both
  - note: *Linux refers to a thread as a task*
- to take full advantage of this you would have to completely understand the behaviour of your application and how it should operate
  - what kernel & OS info should be shared between processes
  - what kernel & OS info should be shared between tasks (threads)
- Linux adds a new system call to create threads and/or processes

```
pid = clone(function, stack_ptr, sharing_flags, args);
```

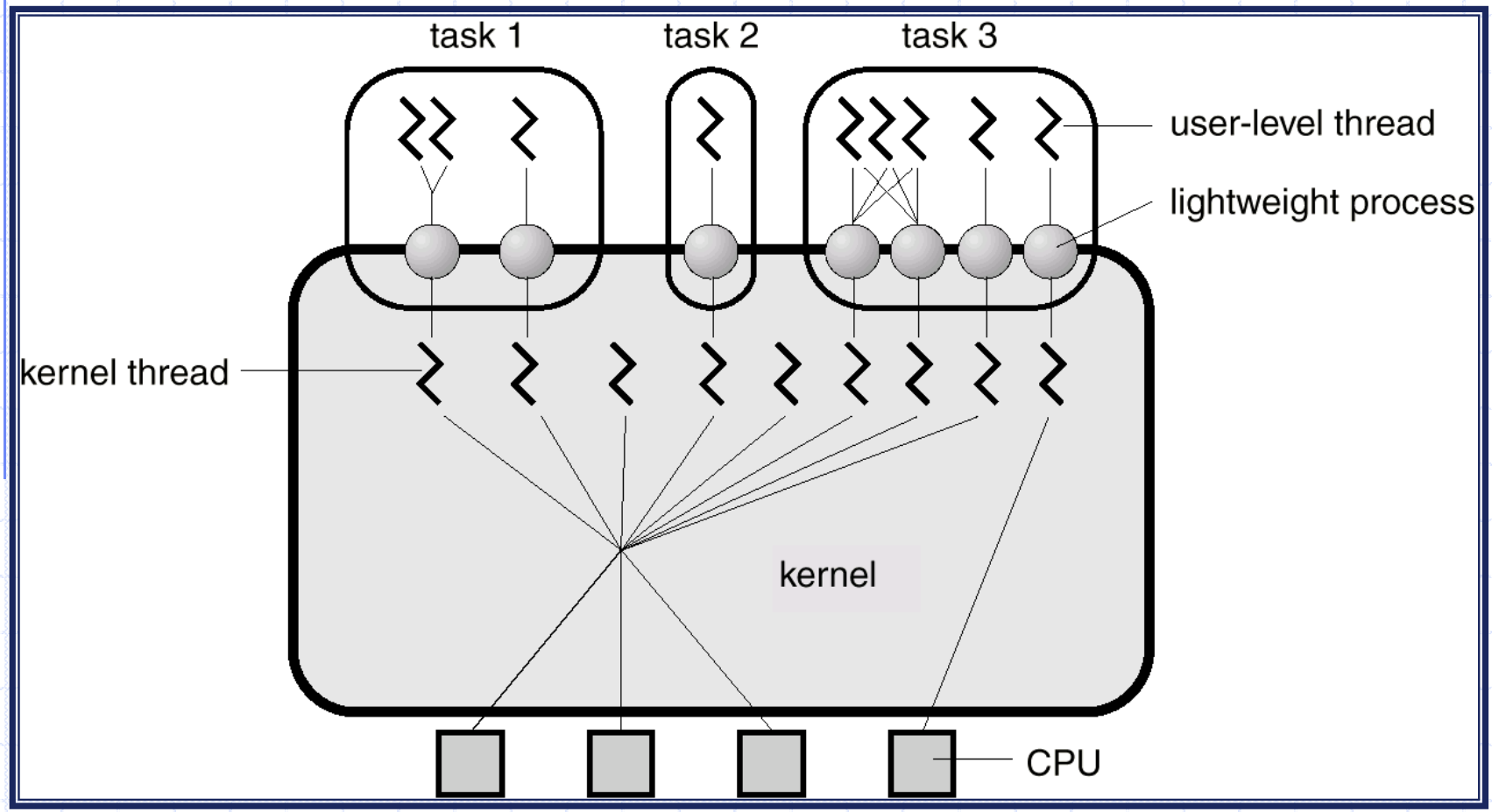
- *by setting sharing_flags appropriately you can create a new process, a new thread, or some hybrid entity*

Linux Threads - 2

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

- threads are supported in the kernel
- each thread will have a PID as well as a task ID
- there is a lot of flexibility with the above approach - but it will not port easily to other OS's
  - is this good? bad?

Solaris Threads



Java Threads

(from http://www.cs.helsinki.fi/u/vihavain/k03/Java/Javathreads1.html)

- threads are **implemented by the Java VM** (Virtual Machine)
  - their behaviour is *heavily influenced by the underlying operating system* and its characteristics
- the actual scheduling policy is system-dependent,
  - *determined together by the host OS and the VM implementation*
  - scheduling between multiple threads may be *preemptive* using time-slicing techniques, so that activities become (randomly) interleaved
  - **or** it could be *non-preemptive* where each thread must itself voluntarily give turn to others (yield)
- the behaviour of threads is inherently *nondeterministic*
  - ie, the order of execution of threads is not repeatable without explicit synchronization
  - Java *does not* support hard real-time systems programming (absolute timing requirements)

Java Threads - 2

- implementations can be either user-level threads package or package managed by the platform OS kernel
- "*green threads*" are simulated threads within the VM
- green threads exist only at the user-level and are not mapped to multiple kernel threads by the operating system
- "*native threads*" are the threads that are provided by the native OS
- native threads can realize the performance enhancement from parallelism (multiple CPUs)
- Java is naturally multi-threaded and because of this the underlying OS implementation can make a substantial difference in the performance of your application

Question 11

11. Assume you have a multi-threaded Java program running on Linux. The threads in your program will be managed by the JVM, which will map them to Pthreads in the kernel.

a) TRUE

b) FALSE

Justification:

Question 12

12. What does it mean if the CLONE_VM flag is set when you are making a clone() system call on Linux?

a) a new processes should be created

b) copy the file descriptors

c) share the signal handler table

d) new thread gets its own files and resources

e) none of the above

The End