Frequently asked questions from the previous class survey

- Is there a separate stack for a process?
- Threads
  - References to objects, when do they terminate, how many is too many, more threads than cores, code example?
- Cores and memory sharing
- Any way to predetermine order of diverse threads for accesses?
- Fork()-exec() and threads

Topics covered in this lecture

- User- and kernel-level threads
- Thread Models
- Thread Libraries

Going about writing multithreaded programs  

- The key idea is to write a concurrent program — one with many simultaneous activities
  - As a set of sequential streams of execution, or threads, that interact and share results in very precise ways
- Subdivide functionality into multiple separate & concurrent tasks
- Threads let us define a set of tasks that run concurrently while the code for each task is sequential

Going about writing multithreaded programs

- Managing data manipulated by tasks
  - Split to run on separate cores. BUT
    - Examine data dependencies between the tasks
- Threaded programs on many core systems have many different execution paths
  - Which may or may not reveal bugs
  - Testing and debugging is inherently harder

User-level threads
User-level threads: Overview

User threads are invisible to the kernel and have low overhead
- Compete among themselves for resources allocated to their encapsulating process
- Scheduled by a thread runtime system that is part of the process code
- Programs link to a special library
  - Each library function is enclosed by a jacket
  - Jacket function calls thread runtime to do thread management
    - Before (and possibly after) calling jacketed library function.

User level thread libraries: Managing blocking calls
- Replace potentially blocking calls with non-blocking ones
- If a call does not block, the runtime invokes it
- If the call may block
  1. Place thread on a list of waiting threads
  2. Add call to list of actions to try later
  3. Pick another thread to run
- ALL control is invisible to user and OS

Disadvantages of the user level threads model (1)
- Assumes that the runtime will eventually regain control, this is thwarted by:
  - CPU bound thread
  - Thread that rarely performs library call ...
  - Runtime can’t regain control to schedule other threads
- Programmer must avoid lockout situations
  - Force CPU-bound thread to yield control

Disadvantages of the user level threads model (2)
- Can only share processor resources allocated to encapsulating process
  - Limits available parallelism

Kernel Threads
Kernel-level threads: Overview

- Process
  - Thread
  - User space
  - Kernel space
  - Thread table
  - Process table

Kernel threads: Overview

- Kernel is aware of kernel-level threads as schedulable entities
  - Kernel maintains a thread table to keep track of all threads in the system
- Compete system wide for processor resources
  - Can take advantage of multiple processors

Kernel threads: Management costs

- Scheduling is almost as expensive as processes
- Synchronization and data sharing are less expensive than processes
- More expensive to manage than user-level threads

Hybrid thread models

- Write programs in terms of user-level threads
- Specify number of schedulable entities associated with process
- Mapping at runtime to achieve parallelism
- Level of user-control over mapping
- Implementation dependent

The Many-to-One threading model

- User threads
  - Kernel thread
Many-to-One Model maps many user level threads to 1 kernel thread

- Thread management done by thread library in user-space
- What happens when one thread makes a blocking system call?
  - The entire process blocks!

September 13, 2018
Professor: SHRIDEEP PALLICKARA

Many-to-One Model maps many user level threads to 1 kernel thread

- Only 1 thread can access kernel at a time
  - Multiple threads unable to run in parallel on multi-processor/core system
  - E.g.: Solaris Green threads, GNU Portable threads

September 13, 2018
Professor: SHRIDEEP PALLICKARA

The One-to-One threading model

- More concurrency
  - Another thread can continue to run, when a thread invokes a blocking system call
  - Threads run in parallel on multiprocessors

September 13, 2018
Professor: SHRIDEEP PALLICKARA

One-to-One Model:
Maps each user thread to a kernel thread

- Disadvantages:
  - There is an overhead for kernel thread creation
  - Multiple user threads can degrade application performance
- Supported by:
  - Linux
  - Windows family: NT/XP/2000
  - Solaris 9 and up

September 13, 2018
Professor: SHRIDEEP PALLICKARA

Many-to-Many threading Model:
2-level is a variant of this

Many-to-Many

September 13, 2018
Professor: SHRIDEEP PALLICKARA

Two-level

September 13, 2018
Professor: SHRIDEEP PALLICKARA
### Many-to-Many model
- **Multiplex** many user-level threads on a smaller number of kernel threads
- Number of kernel threads may be specific to
  - Particular application
  - Particular machine
- Supported in
  - IRIX, HP-UX, and Solaris (prior to version 9)

### A comparison of the three models

<table>
<thead>
<tr>
<th></th>
<th>Many-to-one</th>
<th>One-to-One</th>
<th>Many-to-Many</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True Concurrency</strong></td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>During blocking system call?</strong></td>
<td>Process Blocks</td>
<td>Process DOES NOT block</td>
<td>Process DOES NOT block</td>
</tr>
<tr>
<td><strong>Kernel thread creation</strong></td>
<td>Kernel thread already exists</td>
<td>Kernel thread creation overhead</td>
<td>Kernel threads available</td>
</tr>
<tr>
<td><strong>Caveat</strong></td>
<td>Use system calls (blocking) with care</td>
<td>Don’t create too many threads</td>
<td></td>
</tr>
</tbody>
</table>

### Thread libraries provide an API for managing threads
- Includes functions for:
  1. Thread creation and destruction
  2. Enforcement of mutual exclusion
  3. Conditional waiting
- Runtime system to manage threads
  - Users are not aware of this

### User level thread libraries
- **No kernel support**
- Library code & data structures reside in user space
- Invoking a library function does not result in a system call
  - Local function call in user space

### Kernel level thread libraries
- Library code & data structures in kernel space
- Invoking library function typically **results in a system call**
Thread libraries provide an API for creating and managing threads.

<table>
<thead>
<tr>
<th>Library code and data structures</th>
<th>User level library</th>
<th>Kernel level library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reside in user space</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Can invocation of library function result in system call?</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

DOMINANT THREAD LIBRARIES (1)

- POSIX pthreads
  - Extends POSIX standard (IEEE 1003.1c)
  - Provided as user- or kernel-level library
  - Solaris, Mac OS X, Linux

- Win32 thread library
  - Kernel-level library

DOMINANT THREAD LIBRARIES (2)

- Java threading API
  - Implemented using thread library on host system
  - On Windows: Threads use Win32 API
  - UNIX/Linux: Uses pthreads

JAVA THREADS
Harnesses the thread model of the host OS

JAVA THREADS EXAMPLE

- We will use a thread to perform summation of a non-negative integer

\[ \text{SUM} = \sum_{i=1}^{N} i \]

- If \( N=5 \), we compute the sum of 0 through 5
  - \( 0 + 1 + 2 + 3 + 4 + 5 = 15 \)

JAVA

- Designed from the ground-up to support concurrent programming
  - Basic concurrency support in the language and class libraries

- Java 1.5 and higher
  - Powerful high-level concurrency APIs
JVMs harness the thread models of the host OS

- Windows XP has a one-to-one model
  - So a thread maps to a kernel thread
- Tru64 UNIX uses the many-to-many model
  - Java threads mapped accordingly
- Solaris
  - Initially, used Green Threads \(\rightarrow\) many-to-one
  - Version 9 onwards: one-to-one model

Creating Threads in Java

1. Create a new class \textit{derived} from Thread
   - Override its \texttt{run()} method
2. More commonly used, Runnable interface
   - Has 1 method \texttt{run()}
   - Create new Thread class by passing a Runnable object to its constructor
3. The Executor interface [java.util.concurrent]
   - Has 1 method \texttt{execute()}

Java Threads: Interrupts

- Invoke \texttt{interrupt()} on the Thread
- Threads must support their own interruption
- An interruptible thread needs to
  1. Catch the \texttt{InterruptedException}
  2. Methods such as \texttt{sleep()} throw this, and are designed to cancel the operation and return
- Periodically invoke \texttt{Thread.interrupted()} to see if it has been interrupted

Java Threads: Joins

- If thread object threadA is currently executing
- Another thread can call threadA.join()
  - Causes current thread to pause execution until threadA terminates
- Variants of join()
  - Specify a waiting period

Using Java Threads [1/3]

```java
class Sum {
    private int sum;
    public int get() {
        return sum;
    }
    public void set(int sum) {
        this.sum = sum;
    }
}
```

Using Java Threads [2/3]

```java
class Summation implements Runnable {
    private int upper;
    private Sum sumValue;
    public Summation(int upper, Sum sumValue) {
        this.upper = upper;
        this.sumValue = sumValue;
    }
    public void run() {
        int sum = 0;
        for (int i = 0; i <= upper; i++)
            sum += i;
        sumValue.set(sum);
    }
}
```
Using Java Threads

```java
public class Driver {
    public static void main(String[] args) {
        Sum sumObject = new Sum();
        int upper = Integer.parseInt(args[0]);
        Thread worker = new Thread(new Summation(upper, sumObject));
        worker.start();
        try {
            worker.join();
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }
        System.out.println( "The sum of " + upper + " is " + sumObject.get());
    }
}
```

POSIX: Thread creation

`pthread_create()`
- Automatically makes the thread runnable without a start operation
- Takes 3 parameters:
  1. Points to ID of newly created thread
  2. Attributes for the thread
     - Stack size, scheduling information, etc.
  3. Name of function that the thread calls when it begins execution

POSIX: Thread joining

`pthread_join()`
- Threads that are not detached are joinable
- Undetached threads don’t release resources until
- Another thread calls `pthread_join` for them
- Process exits
- `pthread_join`
  - Takes ID of the thread to wait for
  - Suspends calling thread till target terminates
  - Similar to `waitpid` at the process level
  - `pthread_join(pthread_self());`?
    - Deadlock!
POSIX: Exiting and cancellation

- If a process calls `exit`, all threads terminate
- Call to `pthread_exit` causes only the calling thread to terminate
- Threads can force other threads to return through a cancellation mechanism
  - `pthread_cancel`: takes thread ID of target
  - Depends on type and state of thread

More info on `pthread_cancel`

- State: `pthread_setcancelstate` to change state
  - `PTHREAD_CANCEL_ENABLE`
  - `PTHREAD_CANCEL_DISABLE`
  - Cancellation requests are held pending
- Cancellation type allows thread to control when to exit
  - `PTHREAD_CANCEL_ASYNCHRONOUS`
  - Any time
  - `PTHREAD_CANCEL_DEFERRED`
  - Only at specified cancellation points

Win32 Threads

- `CreateThread`
  - Security Information, size of stack, flag (start in suspended state?)
- `WaitForSingleObject`
- `CloseHandle`

Using Pthreads (1)

```c
#include <pthread.h>
#include <stdio.h>
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */
```

Using Pthreads (2)

```c
int main(int argc, char *argv[]){
  pthread_t tid; pthread_attr_t attr;
  /* get the default attributes */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid, &attr, runner, argv[1]);
  /* now wait for the thread to exit */
  pthread_join(tid, NULL);
  printf("sum = %d\n",sum);
}
```

Using Pthreads (3)

```c
/**
 * The thread will begin control in this function
 */
void *runner(void *param)
{
  int i, upper = atoi(param);
  sum = 0;
  if (upper > 0) {
    for (i = 1; i <= upper; i++)
      sum += i;
  }
  pthread_exit(0);
}
```
The contents of this slide-set are based on the following references