Virtual Processors

1. UThread
   - A Simple Thread System for C
     - The UThread Interface file (uthread.h)
     - Explained
     - `uthread_t` is the datatype of a thread control block
     - `uthread_init` is called once to initialize the thread system
     - `uthread_create` creates and starts a thread to run specified procedure
     - `uthread_yield` temporarily stops current thread if other threads waiting
     - `uthread_join` joins calling thread with specified other thread
     - `uthread_detach` indicates no thread will join specified thread
     - `uthread_self` is a pointer to the TCB of the current thread

2. Reading
   - Text: 2ed: 12.3
   - Text: 1ed: 13.3

3. Synchronous Disk Read using Threads
   - Create two threads that CPU runs, one at a time
   - one for disk read
   - one for `doSomethingElse`
   - Illusion of synchrony
     - disk read blocks while waiting for disk to complete
     - CPU runs other thread(s) while first thread is blocked
     - disk interrupt resets the blocked read
   - In Java
   - `Future` class
   - `resultFuture` from executing function
   - `get()` method

4. Executor Services in Java
   - Create an Executor Service once at beginning of program
     - to manage asynchronous calls in a pool of threads
   - Example:
     ```java
     class ZotRunnable implements Runnable {
         Integer arg;
         public void run() {
             arg = anArg;
             result = zot(result);
         }
     }
     public Integer call() {
         return zot.call(result);
     }
     }
     ```
   - Advantages:
     - better management of result returned or exception thrown
     - asynchronous calls
     - precise thread management abstracted from application code

5. The Virtual Processor
   - Created: in the 1960s
   - Uses: multiple simultaneous instructions per clock cycle
   - Benefits:
     - increased instruction throughput
     - improved system performance
   - Drawbacks:
     - slower memory access times
     - increased complexity of hardware and software

6. Implementing UThreads: Data Structures
   - `uthread_t` is the thread control block
   - `uthread_self` is a pointer to the TCB of the current thread
   - `uthread_init`, `uthread_create`, `uthread_join`, `uthread_detach`, `uthread_yield` are thread management functions

7. Implementing UThreads: Some Questions
   - What is an asynchronous call?
   - What is a thread?
   - What data structures do we need?
   - What basic operations are required?
Example Code for Thread Switch

```asm
; assume volatile (pusha, %rax)

// 1. Save all registers to stack
// 2. Save stack pointer to TCB
// 3. Restore stack pointer
// 4. Restore registers from stack

// Thread Control Blocks

// Ready Queue

// Stacks

// TCB

// Register File

// Example Code

// Example Code for Thread Switch

// Implementing Thread Yield

// Thread Yield

// thread_yield()

// gets next runnable thread from ready queue (if any)

// puts current thread on ready queue

// switches to next thread

// Example Code

// Example Code for Thread Switch

// Implementing Threads: Thread Switch

// Goal

// implement a procedure switch (T_a, T_b) that stops T_a and starts T_b

// T_a calls switch, but it returns to T_b

// Example ... 

// Requires

// saving T_b’s processor state and setting processor state to T_b’s saved state

// state is just registers and registers can be saved and restored to/from stack

// thread-control block has pointer to top of stack for each thread

// Implementation

// save all registers to stack

// save stack pointer to T_b’s TCB

// set stack pointer to stack pointer in T_b’s TCB

// restore registers from stack

// Multiple Processors

// Processor are

// the physical / hardware resource that runs threads

// a system can have more than one

// Uni-Processor System

// a single processor runs all threads

// no two threads run at the same time

// Multi-Processor System

// multiple processors run the threads

// two threads can be running at the same time

// More about this later, but we have a problem now ... 

// how do we compute the value of our_thread, the current thread’s TCB?

// we need this to yield the thread, for example, to place it on ready queue

// but, can’t use a global variable

// Threads Scheduling

// Problem with round-robin, preemptive, priority scheduling

// some applications require threads to run at a certain time or certain interval

// but, what does round-robin guarantee and not guarantee?

// Real-Time Scheduling

// Problem with real-time, preemptive, priority scheduling

// some applications require threads to run at a certain time or certain interval

// but, what does round-robin guarantee and not guarantee?

// Summary

// Thread

// synchronous "thread" of control in a program

// virtual processor that can be stopped and started

// threads are executed by real processor one at a time

// Threads hide asynchrony

// by stopping to wait for interrupt/avert, but freeing CPU to do other things

// Threads state

// when running: stack and machine registers (register file etc.)

// when stopped: Thread Control Block stores stack pointer, stack stores state

// Round-robin, preemptive, priority thread scheduling

// lower priority thread preempted by higher

// thread preempted when its quantum expires

// equal-priority threads get fair share of processor, in round-robin fashion

// Preemption

// Preemption occurs when

// a "yield" is forced upon the current running thread

// current thread is stopped to allow another thread to run

// Priority-based prevention

// when a thread is made runnable (e.g., created or unblocked)

// if it is higher priority than current-running thread, it preempts that thread

// Quantum-based prevention

// each thread is assigned a runtime "quantum"

// thread is preempted at the end of its quantum

// How long should quantum be?

// disadvantage of too short?

// disadvantage of too long?

// typical value is around 10 ms

// How is quantum-based prevention implemented?