Threads in Java

Create a procedure that can be executed by a thread
• build a class that implements the Runnable interface

Runnable

Implement Threads: Some Questions

The key new thing is blocking and unblocking
• what does it mean to stop a thread?
• what happens to the physical processor?
What data structures do we need

What basic operations are required

Implementing UThreads: Data Structures

Thread State
• running: register file and runtime stack
• stopped: Thread Control Block and runtime stack
• blocked: what happens to the physical processor?

Thread-Control Block (TCB)
• thread status: (NAGENT, RUNNING, RUNNABLE, BLOCKED, or DEAD)
• pointers to thread's stack base and top of its stack
• scheduling parameters such as priority, quantum, pre-emptability etc.

Ready Queue
• list of TCB's of all RUNNABLE threads
• one or more Blocked Queues
• list of TCB's of BLOCKED threads

TCB Control Blocks

The Virtual Processor

Originated with Edsger Dijkstra in the THE Operating System
In the Structure of the “THE” Multitasking Operating System, 1965
I had had two previous experiences (dating back to 1960) in making basic software dealing
with multitasking, and I knew better than experience that a result of the impracticality of the
interrupt-driven system was that programs could not present ideas in a way that
was possible and prevented loss better than later, try to prevent ready delays
from entering the construction.
This decision is made at the bottom of what I regard as the group's main
contribution to the art of system design.

The Process (what we now call a Thread)
• a single thread of synchronous execution of a program
• the illusion of a single process such as the Single Machine
• can be stopped and restarted
• stopped when waiting for an event (e.g., completion of an I/O operation)
• restarted with the event fires
• can co-exist with other processes sharing a single hardware processor

Executor Services in Java

Create an Executor Service
• to manage asynchronous calls in a pool of threads (here limited to 2)

ZotCallable (Integer anArg) {
  // do something with anArg
  return result;
}

Future<Integer> resultFuture = ex.submit (
  ZotCallable (Integer anArg)
);

result = resultFuture.get ();

// synchronizes with main thread, returns the result

Example Program using UThreads

print printf ();

畅通的程序

Thread Data Structure Diagram

Thread Status DFA

Ready Queue
Thread Control Blocks
Stacks

Nascent
Running
Blocked
Ready
Join
Detach
Thread Create
Thread Join
Thread Yield
Thread Block
Thread Unblock
Implementing Threads: Thread Switch

**Goal**
- implement a procedure switch (Ta, Tb) that stops Ta and starts Tb
- Ta calls switch, but it returns to Tb

**Requires**
- saving T_a'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_a's TCB
- set stack pointer to stack pointer in T_b's TCB
- restore registers from stack

**Thread Switch**

1. Save all registers to A's stack
2. Save stack top in A's TCB
3. Restore B's stack top to stack-pointer register
4. Restore registers from B's stack

**Thread Private Data**

- threads introduce need for another type of variable
  - a thread-private variable is a global variable private to a thread
  - like a local variable is private to a procedure activation
- for example
  - cur_thread, the address of the current thread's activation frame
  - it's a global variable to thread, but every thread has its own copy

**Implementing Thread Private Data**
- store thread-private data in TCB
- store pointer to TCB at top of every stack
- compute current stack top from stack pointer
- require that stack top address is aligned to stack size
- stack top = r5 & ~(Stack Size - 1)

**Preemption**

- preemption occurs when
  - a "yield" is forced upon the current running thread
  - thread with highest priority goes first
- priority-based preemption
  - 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 preemption
  - 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 100 ms
- how is quantum-based preemption implemented?

**Real-Time 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
  - hard real-time – e.g., for controlling or monitoring devices
    - thread is guaranteed a regular time slice and is given a time budget
    - thread can not exceed its time budget
  - soft real-time – e.g., for media streaming
    - option 1: over-provision and use round-robin
    - option 2: thread expresses its scheduling needs, scheduler picks its best, but no guarantee

**Summary**

- notice that we haven't talked about the OS yet (we will soon)
- everything we've talked about can be implemented in an application
- the difference between OS and application is processor privilege level
- OS is "kernel"-level
- applications are "user"-level
- and so, what we are talking about are called User-Level Threads

**User-Level Threads**

- when running, stack and machine registers (register file, etc.)
- when stopped: Thread Control Block stores stack pointer, stack stores state

**Thread State**

- when running: highest priority thread
- when stopped: Thread Control Block stores stack pointer, stack 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

**Preempted** when its quantum expires
- equal-priority threads get fair share of processor in round-robin fashion