CS 350 Notes

Matthew Visser

Sep 20, 2011

1 Threads

1.1 Thread Interface

- Fork – create new
- Exit – destroy
- Yield – let another thread run, but reschedule at some point.
- Sleep – wait for an event
- Wake-up – make threads sleeping on an address wake up.

1.2 Fork

Forking creates a new thread. It gives

- A name – for debugging
- a function to execute – has 2 arguments

This function needs to

1. Allocate a stack
2. Zero out the stack, store a pointer to the thread function on the stack's thread context. Stored at address `40(sp)`.
3. Add thread to thread library. Create a `struct thread` for the new thread in the thread library and store `sp` in the thread structure.
4. Extra:
   (a) Pass parameters to thread function

(b) Call `thread_exit` when thread is done.

1.3 Scheduling Threads

- FIFO – maintains a queue of threads that are ready to be scheduled. When a thread yields or forks, it's moved to the back of the queue. Queue is stored in kernel data segment in library.

1.3.1 Preemption

- `Yield()` allows programs to voluntarily pause
- Sometimes it is desirable to make a thread stop running even if it has not called `Yield()`.
- an involuntary context switch is called *preemption* of the running thread.
- To implement preemption, the thread library must have a means of taking over control, even though the application has not called a thread library function
- This form of taking control is normally called *interrupts*.
- Easier to write code for a preemptive scheduler – don't have to worry about yielding CPU. Makes the job of a programmer simpler.

1.3.2 Interrupts

- An event that occurs while a program executes
- caused by system devices: timer, disk controller, network interface
- When an interrupt occurs, hardware automatically transfers control to a fixed location
- At the memory location, the thread library places a procedure called an *interrupt handler*
- The handler normally
  1. Saves current thread context. This is saved on the stack of the *current* thread.
  2. determines what device caused interrupt and performs specific processing
  3. restores saved thread context and resumes execution

1.3.3 Round-Robin Scheduling

- One example of preemptive scheduling.
- Similar to FIFO, but is preemptive
- Gives each thread a fixed time to execute before it is preempted.
- The time it executes is called the *scheduling quantum*.

To implement:

- Suppose the timer generates an interrupt at t time units
- suppose the thread library wants a quantum $q = 500t$
- To implement this, the thread library can maintain a variable called `running_time` to track how long a thread has been running
  - Reset to 0 when thread dispatched.
  - When an interrupt occurs, the timer-specific code can increment `running_time` and test its value, invoking `Yield()` if the time is equal to q.
- See slide 23 of `threads.pdf` to see a diagram of stack.

2 Synchronization

2.1 Concurrency

- Threads run concurrently, share access to data, can run at the same time.
- Need to enforce *mutual exclusion*.
- The part of the program where a shared object is accessed is called the *critical section*.

2.1.1 Mutual Exclusion

- Can ensure only one thread executes a critical section at a time.
- Server techniques to enforce
  - Exploit special hardware specific instructions
  - Use algorithms that rely on atomic loads and stores
  - Control interrupts to ensure threads are not preempted while they are executing a critical section

2.1.2 Disabling interrupts

On a uni-processor, only one thread is running at a time, so we can just disable interrupts.

If the running thread enters a critical section, mutex can be violated if

1. The thread is preempted
2. The scheduler chooses a different thread to run

Since preemption is caused by timer interrupts, mutex can beenforced by disabling interrupts before a thread enters a critical section, and re-enabling them when the thread leaves the critical section.

This is how OS/161 does this. It calls `splhigh()`.

Pros:

- Doesn't require hardware-specific synchronization instructions
- Works for any number of threads

Cons:

- Indistriminate: prevents all preemption.
- Ignoring timer interrupts has side effects.
- Will not enforce mutex on multi-processor.

2.1.3 Priority Levels in MIPS

Has levels from $0 \rightarrow 15$. From 0 to 14, interrupts are enabled. `splhigh()` means to disable interrupts completely. The level is 15, and no other thread is higher priority.