CS 350 - Operating Systems **Winter 2018**

Lecture 2: January 10, 2018

Lecturer: Lesley Istead *Notes By: Harsh Mistry*

2.1 Threads and Concurrency Continued

2.2 Implementing Concurrent Threads

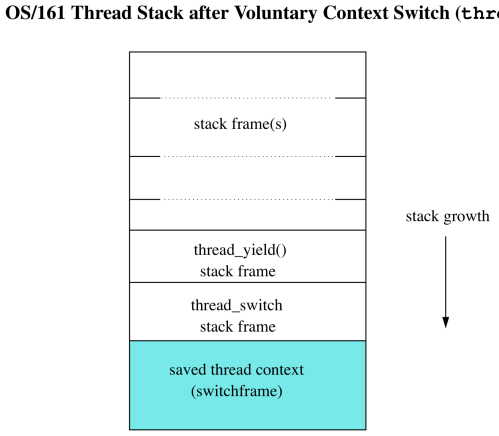
- Option 1 : multiple processors, multiple cores, etc
- Option 2 : time sharing
  - multiple threads take turns on the same hardware
  - rapidly switch from thread to thread so that all make progress

2.3 Time-sharing and Context Switches

- When timesharing, the switch from one thread to another is called a **context switch**
- During a context switch :
  1. Decide which thread will run next
  2. Save registers contents of current thread
  3. Load register contents of next thread
- Thread context must be save/restored carefully, since thread execution continuously changes the context.

2.3.1 What Causes Context Switches?

- Calls to `thread_yield`
- Calls to `thread_exit`
- Blocks via call to `wchan_sleep`
- **preempted** threads
  - runnign thread **involuntarily** stops running



12 Diagram taken from class slides

2.3.2 Preemption

- **Preemption** prevents a running thread from potentially running forever
- **Preemption** means forcing a running thread to stop running, so that another thread can have a chance.
- To implement preemption, the thread library must have a means of "getting control" even though the running thread has not call a thread library function. This is normally accomplished using **Interrupts**

2.3.2.1 Interrupts

- An interrupt is an event that occurs during the execution of a program
- Interrupts are caused by system devices
- When an interrupt occurs, the hardware automatically transfers control to a fixed location in memory
- At that memory location, the thread library places a pricedure called an **Interrupt handler**
- The interrupt handler normally :
  1. Creates a **trap frame** to record context at the time of the interrupt
  2. Determines which device caused the interrupt and performs device-specific processing
  3. Restores the saved thread context from the trap frame and resumes execution of the thread.

2.3.2.2 Preemptive Scheduling

- A preemptive scheduler imposes a limit, called the **scheduling quantum** on how long a thread can run before being preempted.
- The quantum is an **upper bound** on the amount of time that a threa can run. It may block or yield before its quantum has expired.

- Periodic timer interrupts allow running time to be tracked.
- If a thread has run too long, teh timer interrupt handler preempts the thread by calling `thread_yield`
- The preempted thread changes state from running to ready, and its is placed on the ready queue.

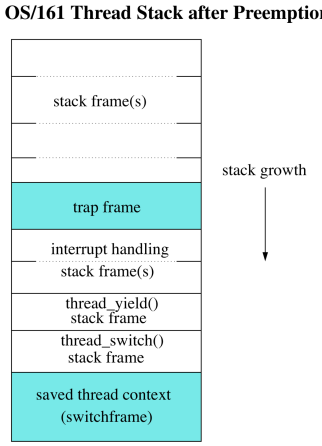


Diagram taken from class slides