Operating Systems
COMS W4118
Lecture 14

Alexander Roth

2015 – 03 – 10

1 How to Implement Locks

1.1 Disable Interrupts

- Used in early linux kernel when there was only one linux processor.
- As soon as there were multiple CPU's this method stopped working.

2 Implementing Locks Using Software

- Peterson's algorithm works, but not in modern systems.

2.1 First Attempt

- Most naïve attempt; there is an integer flag and you can check if it is set.
- When the flag is zero, someone has unlocked the lock.
- This method does not work as there will be a gap between the time you set the lock and the time you check the lock.
- It is possible that both threads could enter a critical section without realizing there is another thread.
- Any time you see a lock that needs a flag to be set, you need to keep an eye out for a gap in the program where two threads could occur.

2.2 Second Attempt

- Two flags, both set to 0.
- First flag mapped to the first thread and so on.

- When a thread wants to take a lock, it calls `lock`.
- It then sets the corresponding flag number.
- It checks the flag previous to it to see if the other thread wants to be there.
- As long as the other thread does not want the lock, the current thread can go forward.
- You set your flag and then you test the other flag; thus, it eliminates the critical section race condition.

2.3 Third Attempt

- Single variable that indicates who's turn it is.
- When it is your turn, you run. Otherwise, you wait.
- Achieves mutual exclusion, only one thread can run at a time.
- There is no way that two threads can be in a critical section together.
- The problem is that it can create a deadlock.
- Three conditions for a lock"
  1. Must provide mutual exclusion
  2. Must have each thread progress.
  3. There cannot be starvation on a thread.
- Mutual exclusion is preserved.
- Thread progress is not preserved.
- What happens if a thread enters a critical section, grabs a lock, unlocks, and wants to lock again.
- However, when you unlock, you must give away your turn.
- A lock cannot depend on the thread's behavior.
- This system depends on threads outside of the critical section of the system.

2.4 Final Attempt: Peterson's Algorithm

- Combination of the previous two approaches.
- Turn-based mechanism AND intent-based mechanisms.
- Problem with turn-based system was that you must alternate.
- Here you kind of do the same thing, but first you wait until its the other guy's turn.
- It only matters who's turn it is as long as both are interested in using the system.
- If both are not interested, then you don't have to wait for the other thread's turn.

2.5 Memory Barriers

- Ensures that all memory operations up to the barrier are executed.
- Peterson algorithm will not work because of modern CPU behavior.
- Barrier is used a lot in Linux code.

3 Version 3: Hardware Instructions

- Version 1: Disable interrupt
- Version 2: Software Locks
- The only practical solution is to have a hardware instruction.
- Have a static variable called `flag`, when `test_and_set` operates, return the previous value of the flag.
- Busy waiting is bad.
- `volatile` keyword means that the memory location pointed to by the pointer may change behind you.
- So multi-threaded programs need to know that these memory locations change.

4 Spin-wait or block?

- Problem of spin-wait: waste CPU cycles.
- You want to avoid sleeping as a sleep command has an overhead attached with it.
- Having a spin lock is good for short operations.
- You only spin when the lock is taken by another thread.
- Spin locks should be used if the lock is released quickly as long as the critical section is small.
- It the thread has a large critical section, it is better to put the thread to sleep and check after the thread is done.
- Typically not used for user-programs as user programs can be preempted at any time.
- Spin-locks are not useful when you cannot control the preemption of the system.
- POSIX has a spinlock API, but it is rarely used.
- If you are using a uni-processor, it's a bad idea to use a spin-lock becuase it will spin forever.
- There is no reason to spin if you are not going to get interrupted.

5 Yield

- Instead of continuously checking a process, just put the process to sleep and check at a later time.
- The operating system will move onto a different process.
- The issue is that someone has to wake up the threads to check for the flag.
- **Thundering Herd** - all the locks wake up to just have one lock receive and then the rest will go back to sleep.
- These processes are just very inefficient; why should all the threads wake up when only one thread will take the event.