10/4/10

How to approach a system call

Don't start thinking about writing code.

DO think about the purpose of a system call
- "Safe" mechanism by which user programs are able to access kernel-protected resources

Lock class & Condition are kernel resources

Lock & Condition system calls are just interfaces to Lock & Condition kernel objects

Example: Acquire

Task: Provide access to a Lock object

* Locks are specific to a process *

The Kernel must have a real Lock object - created upon request by a user program. - CreateLock syscall

Locks must be retrievable for Acquire, Release, Destroy Lock syscalls

Condition variables require a lock object to be passed in as an argument

Format of syscalls from user program viewpoint

What this means:

1. What data does the OS need to carry out the request
2. Is there a return value to the user program?

int CreateLock
int CreateCondition
} Making a resource for the user program for later use

- Must return a "handle" to the user program - a resource identifier

Let's use an array (kernel) to store created Locks & CVs

- 1 array for each - 1 for Locks, 1 for CVs

My kernel return value is the index position in the array of the new object

Because of my choice of an array, I have to track of where the next index position is for the next new resource

- how do I know when I run out of space in my array

* - Use a simple counter.
  - Every time a Create syscall is requested I increment the counter
  - Be sure to check for a full array

The tables & counters are shared by all threads issuing system calls

- Need a kernel lock - used by the kernel - to protect these shared kernel resources

The array cannot be an array of Locks/Conditions

- We cannot enforce the process-specific rule

- Need some piece of data in the array so the kernel can enforce the process rule
  - If you have a unique PID in your process table you can use it
  - Or... AddrSpace pointer

For now...

```
struct LockEntry {
    Lock *lock;
    AddrSpace *ownerProcess;
};

LockEntry lockTable[MAX_LOCKS];
```

One last Key Point: DestroyLock

Task: Delete an existing Lock

Issue: One thread, in a process, issues a DestroyLock ~~before~~ ~~the~~ while the Lock is in use

OS Help: Only destroy a resource when it is not in use.

What does it mean for a Lock to be in use?
- state should be "busy"

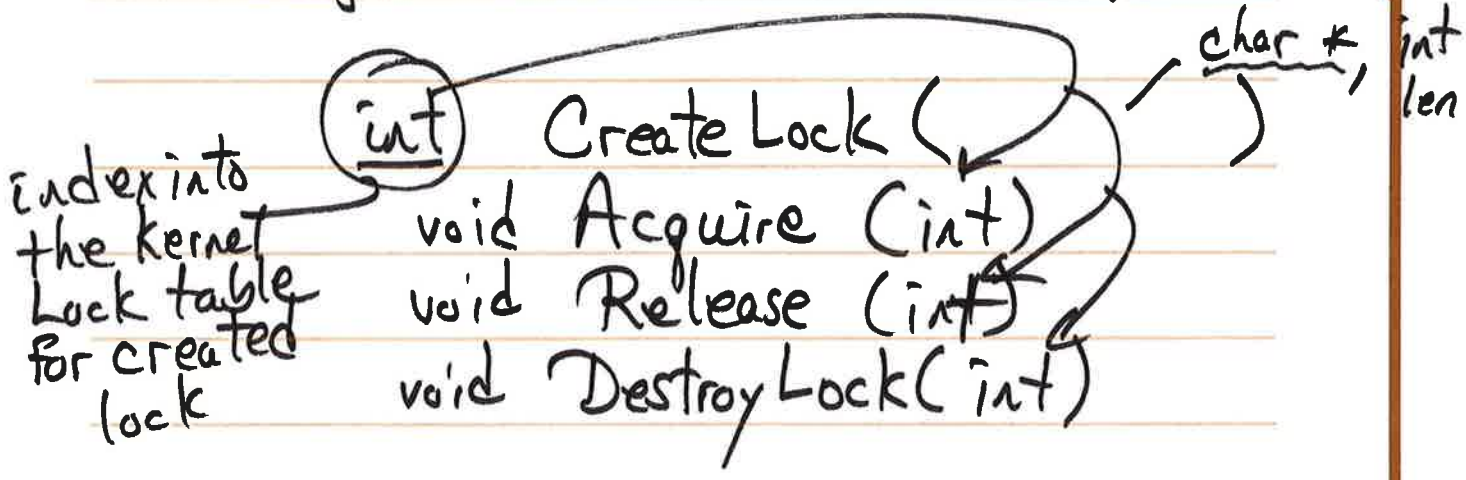
Rule: IF I (OS) get a DestroyLock request, I only delete the lock when the lock state is "available"

A new issue: If we don't "remember" a DestroyLock has been requested, the user program would be forced to issue - worst case - multiple DestroyLocks to delete a lock.

new

```
struct LockEntry {
    Lock* lock;
    AddrSpace* ownerProcess;
    bool toBeDeleted = false;
};
```

Signature of all Lock syscalls




```
int  CreateCondition( char *, int len )
void Wait   ( int cond#, int lock# )
"    Signal   ( int cond#, int lock# )
"    Broadcast( int cond#, int lock# )
void DestroyCondition( int cond# );
```

syscall.h has:

* void Halt(); function declaration

To the compiler: There is a function called Halt

In my user program, I can
#include "syscall.h"

```
{
int main() {
    Halt();
    return 0;
}
}
```

① compile ⇒ object file

② Link ⇒ links all object files w/ any other libraries to make an executable

There is C/C++ function called Halt() - it is just a "stub"

The connection is in start.s. There is assembler code that "converts" the stub function into a jump to Nachos Kernel

The pattern for syscall implementation

① Get the user program arguments

```
int ___ = machine->ReadRegister(4)
                                5
                                6
                                7
```

② Validate the arguments

If any argument is invalid, print an appropriate error msg and return

③ Perform the task

④ If called for, return a value

Acquire

lockTableLock -> Acquire();

- validations

Lock* lock = lockTable[index] -> lock;

isToBeUsed = true;
isInUse

Do the task → lock -> Acquire();

lockTableLock -> Release();

Need a status variable that a lock is going to be used

Scenario: A lock is not in use, BUT the thread next in the Ready Q has an Acquire about to be executed

The thread in the CPU has requested a Destroy Lock
- ~~before~~ after validation this thread is context switched out

This cannot be allowed to happen

DestroyLock
lockTableLock->Acquire();
- validation

context switch →
- "delete" the entry
lockTableLock->Release();