Nick Passantino

CS360

Professor Serce

3/11/19

**Lock Implementation is OS161**

lock struct

The lock struct has several variables and structs (although since the structs have associated functions I will refer to them as objects) within it. The first variable it has is a char pointer variable named lk\_name, which is a name associated with the lock that is useful for debugging purposes. The lock also has a spinlock object, which is used to ensure some sections of the lock’s code do not have more than one thread accessing them, and is also used by the waiting channel. The waiting channel (wchan) is an object used by the lock to block threads (put them to sleep) that try to acquire the lock while it is being used.

The threads that try to acquire a lock in use are put in the waiting channel and are blocked until the lock is released. The last item that the lock struct contains is a volatile thread pointer, which is a pointer which holds the location of the thread that holds the lock. The pointer is volatile because its value can change irrespective of the code immediately surrounding it (at any time and by multiple threads).

lock\_create()

This function initializes all the things that the lock needs and returns a pointer to the created lock. First, the space for the lock is allocated and the lock is given a name (which is passed into the function). Then the lock’s holding thread is set to NULL while its spinlock and waiting channel are initialized. If the lock is unable to create a wchan, it deallocates the objects and returns null, otherwise a pointer to the newly created lock is returned.

lock\_destroy()

Lock\_destroy deallocates all of the lock’s objects and variables. The spinlock and the wchan have special cleanup functions that must be called before the rest of the lock is deleted.

lock\_acquire()

This function allows the caller to acquire the lock. Since only one thread may possess the lock at a time, and multiple threads should not try to possess the lock at the same time, a spinlock is set to encapsulate the lock-acquire code. Next, because interrupts would cause errors in this function, the Priority Level is set to its highest. The function then has a while loop, which checks to see if the lock is currently held (the lock’s holding\_thread pointer is not NULL). If it is available, the thread can move on to take the lock. If the lock is being held, the thread is put into the waiting channel.

lock\_release()

After checking that the calling thread currently holds the lock, the function sets the PL to its highest setting (to disable interrupts) and sets the lock’s holding\_thread pointer to NULL. The waiting channel is then notified to wake up a thread that is asleep. The lock-release code is surrounded by a spinlock to ensure that multiple threads do not try to release the lock at the same time (potentially waking up multiple threads).

lock\_do\_i\_hold()

This function returns true when the calling thread is the thread that holds the lock. This is checked by comparing the lock’s holding\_thread pointer to curthread, a pointer that points to the thread that is currently being run.