Luke Pepin – CSE 4300

Assignment 4

Released: March 13, 2024

Due: Apr 5, 2024 11:59PM

Explanation: The existing synch.c and synch.h files contain implementation for basic synchronization primitives. However, to properly implement locks they need more advanced synchronization primitive first locks and later condition variables. By adding to these files, it is possible to integrate these requirements into the OS. To test if the implementations were done correctly the given test files of sy2 and sy3 were run and their output screenshotted.

Notes: The spacing of the document is a little strange because I wanted to ensure my explanations and their represented screenshots were on the same page. Please also note the screenshots of the code were done in my coding environment Visual Studio Code not the Virtual Machine the code however was tested in the virtual machine.

Synch.c:

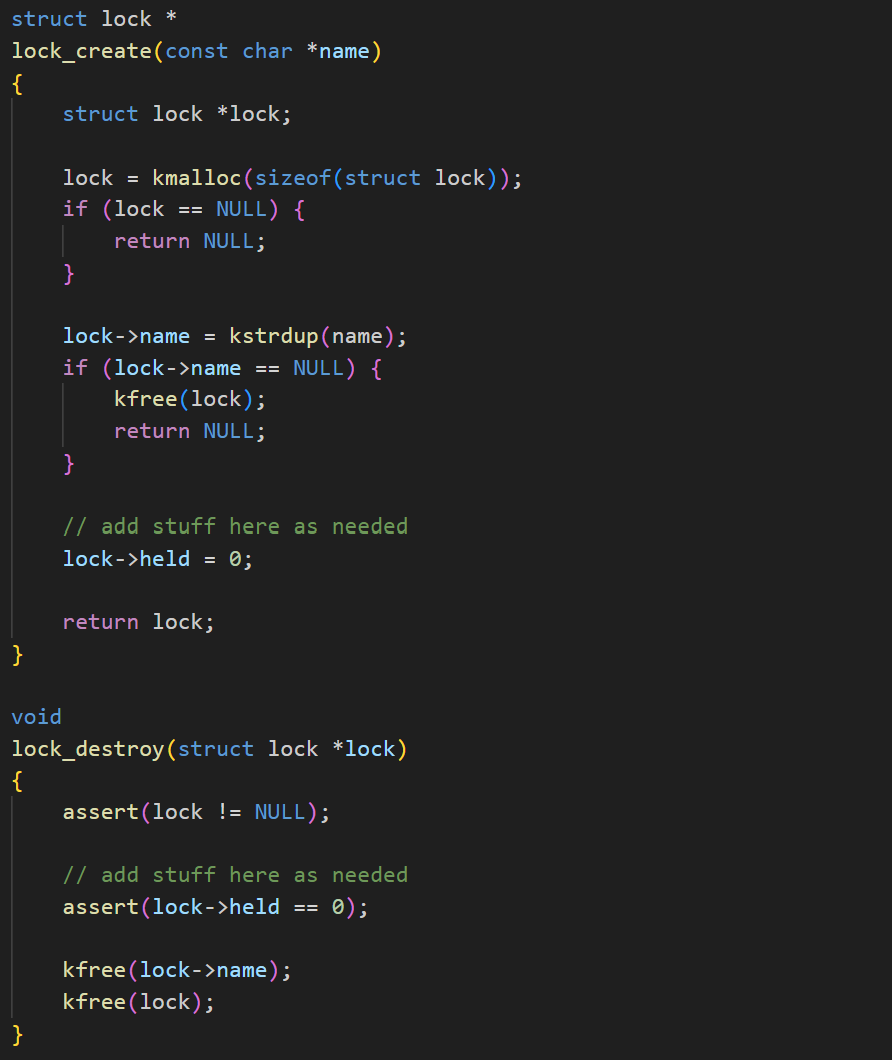
**Explanation of Changes:**

**Locks:**

lock\_create: added lock->held = 0; to track whether the lock is currently held or not

lock\_destroy: added assert(lock->held == 0) to ensure the lock is not held before destroying it

**lock\_create\_and\_destroy\_changes.png**

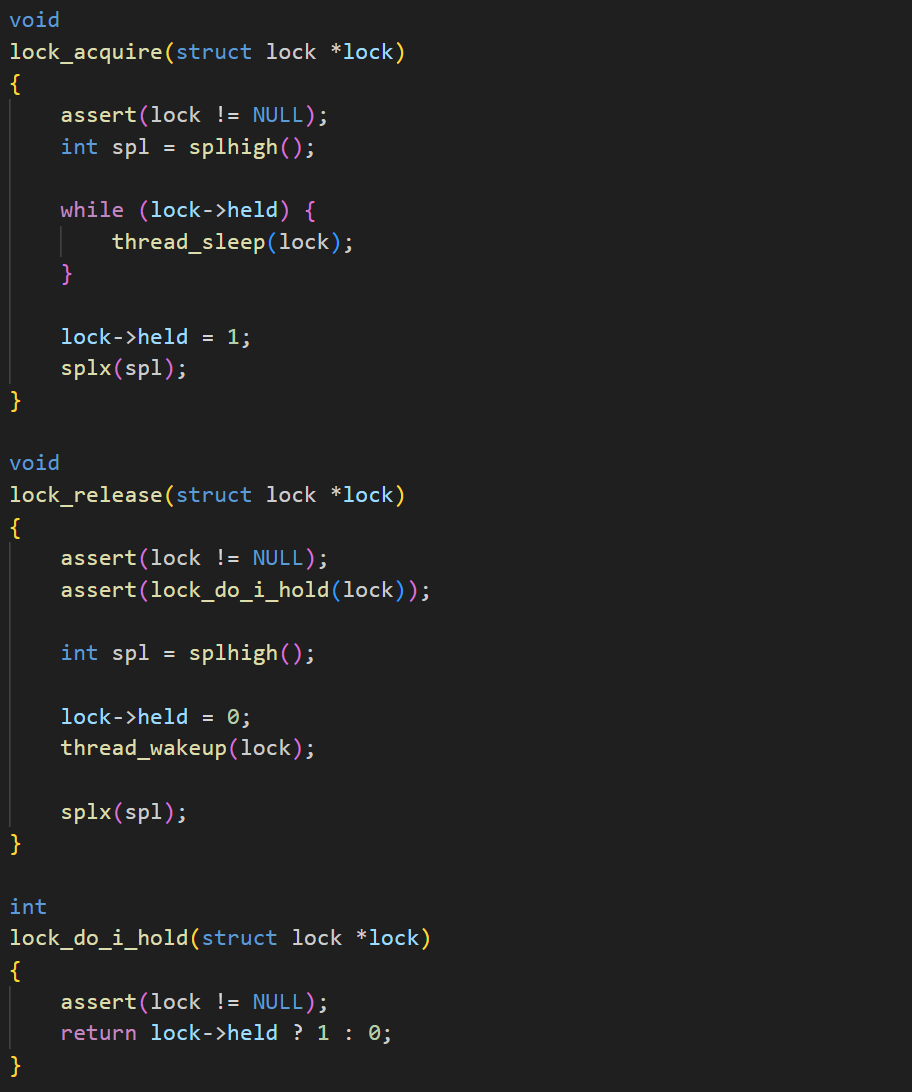


lock\_acquire: implemented a sleep lock which sleeps until the lock is avaible and then sets lock-> held = 1;

lock\_release: implemented a way to release the lock if it passes all the requirements then sets lock->held = 0; and wakes up one thread waiting on the lock.

lock\_do\_i\_hold(): checks if the current thread holds the lock, this function is used in many other functions

**lock\_functions\_changes.png**

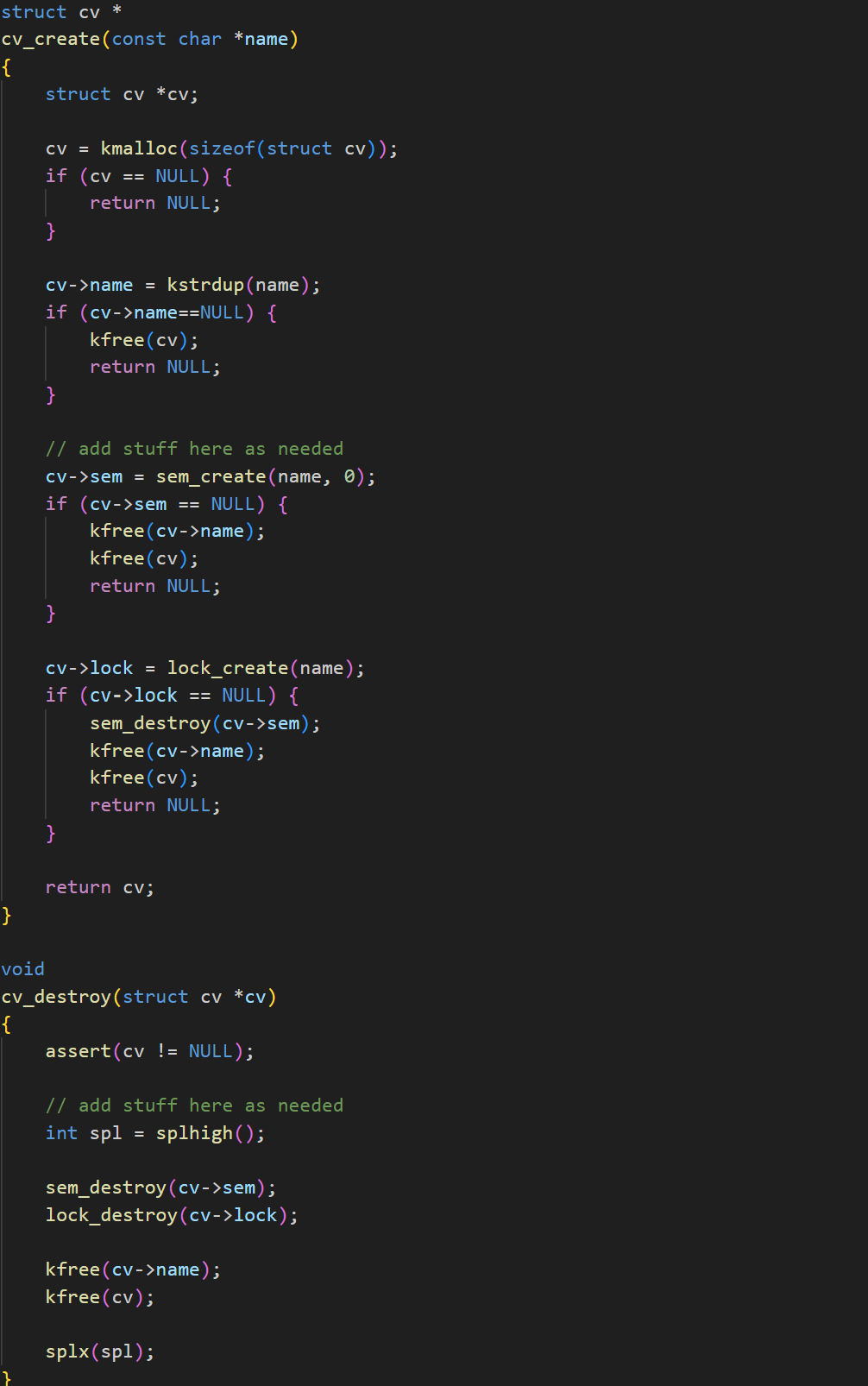


**Condition Variables:**

cv\_create: added sem and lock creation and initialization, as they are new fields of the cv structure and required for the later functions

cv\_destroy: added lines to destroy sem and lock as well as splhigh() and splx(spl) to manipulate the order of threads and their competition.

**cv\_create\_and\_destroy\_changes.png**



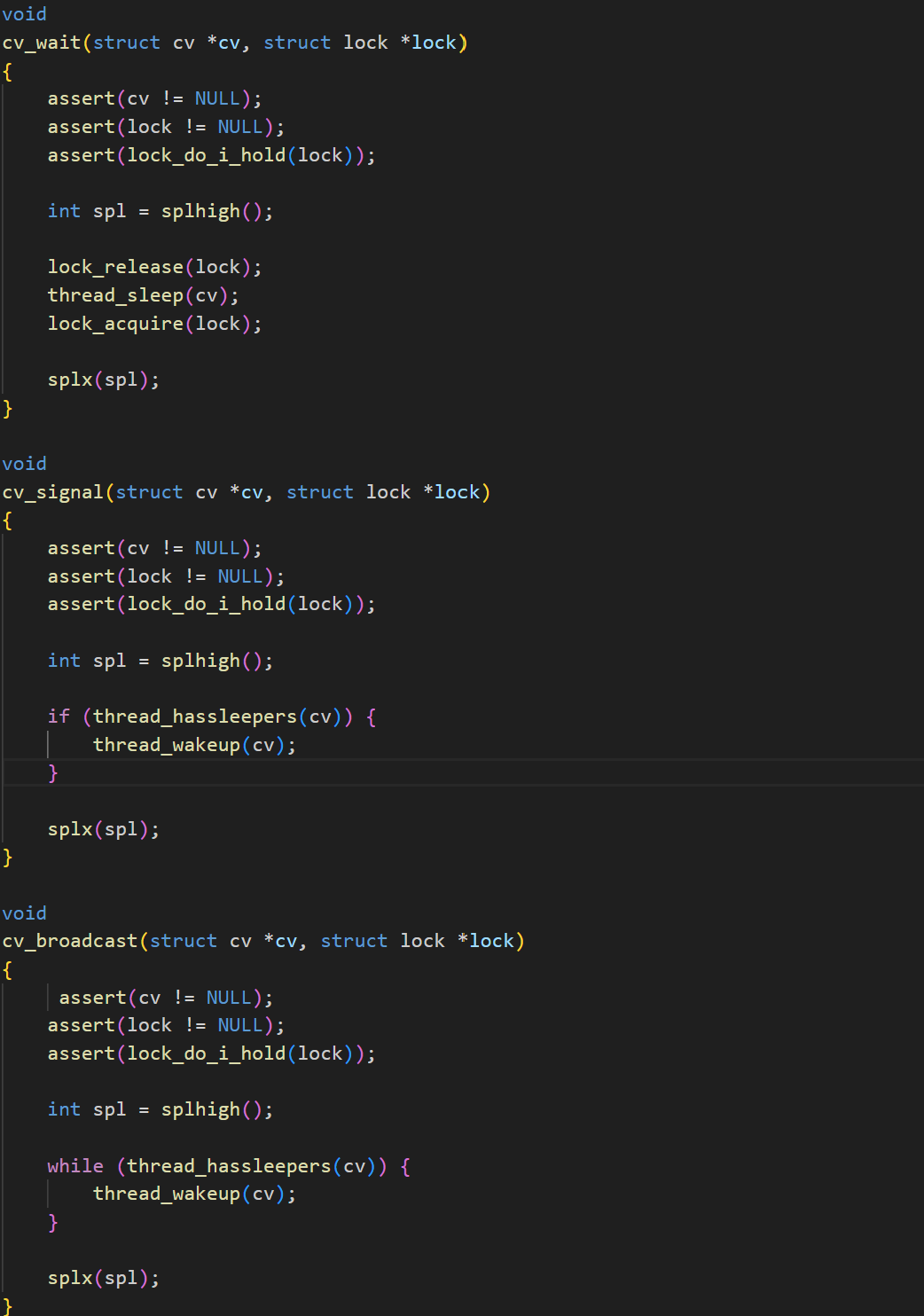
All 3 functions have splhigh() and splx(spl) lines to manipulate the final order of the threads:

cv\_wait: implemented a way to release the lock, put the thread to sleep on the cv and then re-acquire the lock after waking up

cv\_signal: implemented a way to wake up one thread waiting on the condition variable

cv\_broadcast: implemented a way to wake up all threads waiting on the condition variable.

**cv\_function\_changes.png**



Synch.h

**Explanation of Changes:**

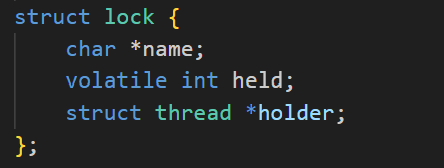
2 new fields were added to each structure to work with the new synch.c

**struct lock**:

Added volatile int flag; which indicates if lock is held (1) or not (0)

Added struct thread \*holder; indicates the thread that currently holds the lock

**Struct\_lock.png:**

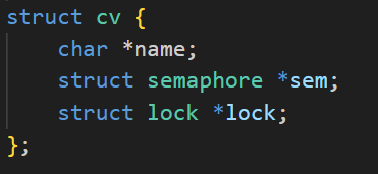


**struct cv:**

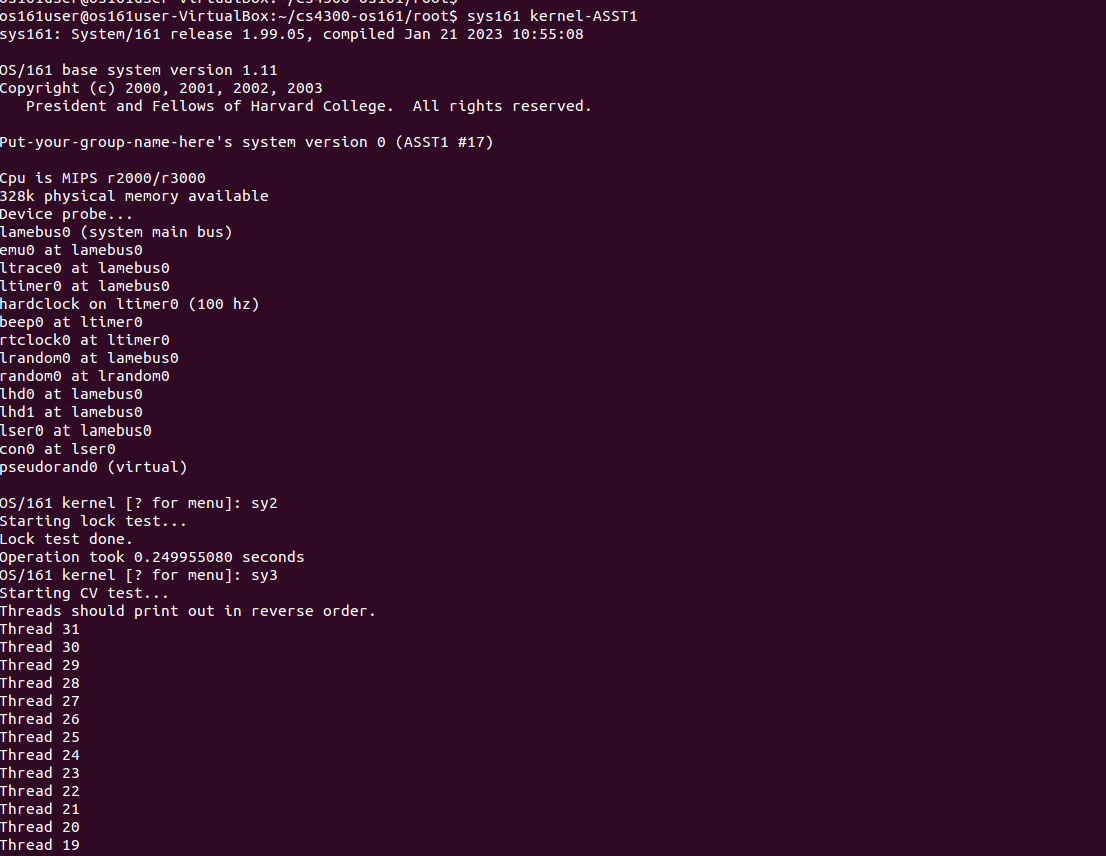
Added struct semaphore \*sem; indicates a point to a semaphore struct

Added struct lock \*lock; indicates the associated lock with the cv

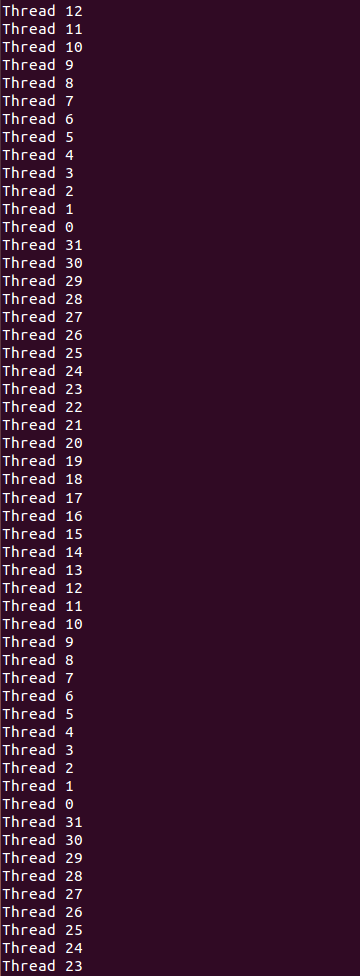
**Struct\_cv.png:**



**sy23\_completion.png** (*Please note I had to restart my VM from scratch, so my name is not present before system version 0)*



Continued:





The thread number is printed 5 times each to show the threads complete some work in their lifecycle properly continuing to the next thread and can continue again afterwards.

This could be a bug resulting from many factors, however 5 seems like a human chosen number after multiple runs of the code it continually does it 5 times.