**Lab #3: xv6 Threads**

Submitted by: Priyanka Jadli (862325015) and Chirag Rajavat (862324954)

# **Demonstration link**

<https://youtu.be/O50_okJV9Yc>

# **List of files modified**

1. kernel/defs.h
2. kernel/proc.c
3. kernel/syscall.c
4. kernel/syscall.h
5. kernel/sysproc.c
6. user/frisbee.c
7. user/thread\_lib.c
8. user/user.h
9. user/usys.pl
10. Makefile

# **Modification screenshots**

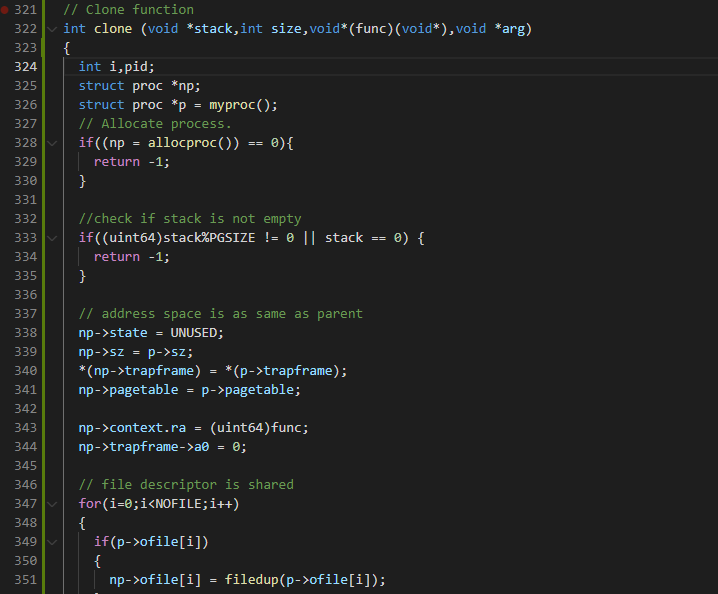
1. kernel/defs.h

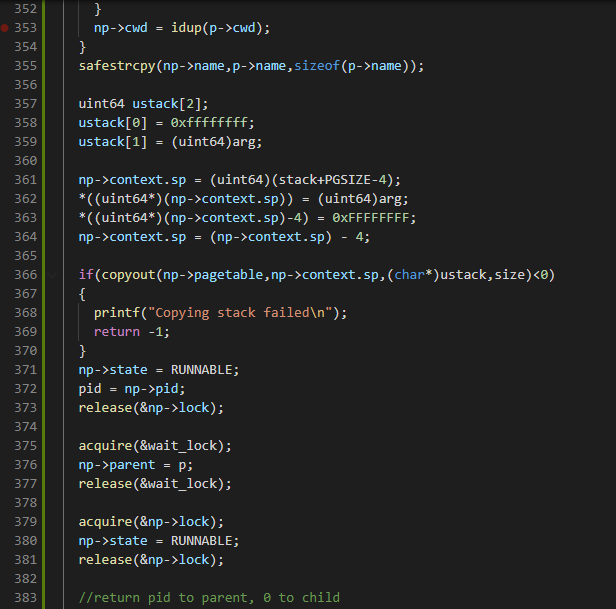
Added entries for kernel functions

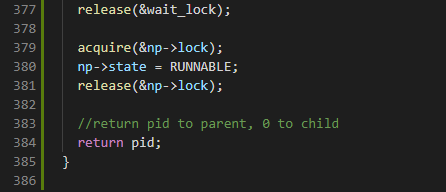


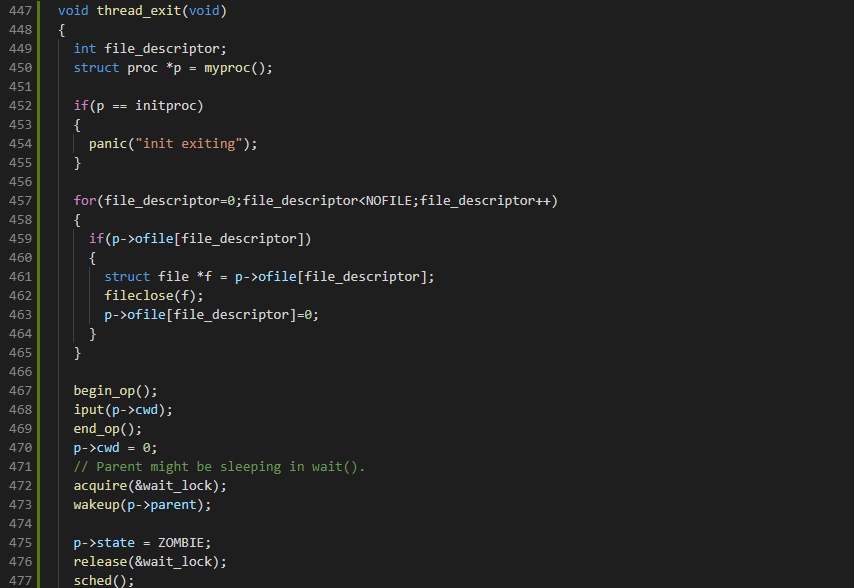
1. kernel/proc.c

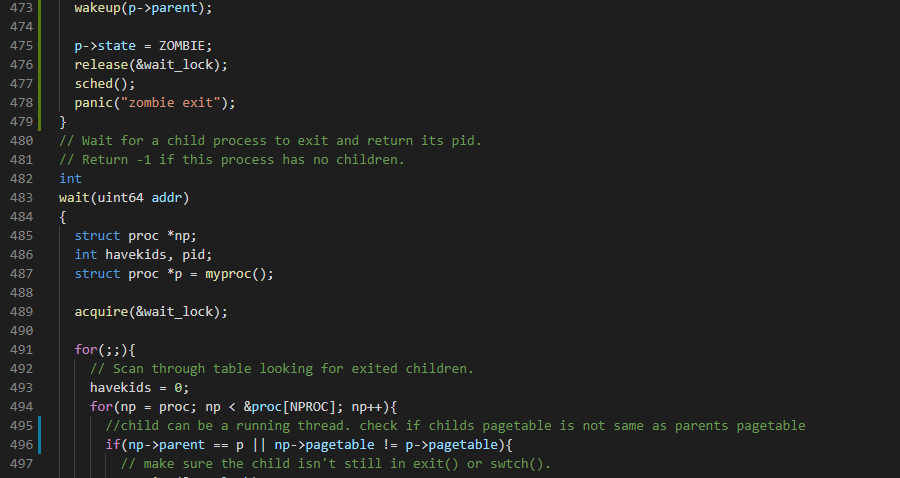
Implemented function clone() to create threads. Create function thread\_exit() for exiting threads. Modified function wait().











1. kernel/syscall.c

Added declaration for our both system calls sys\_clone and sys\_thread\_exit.





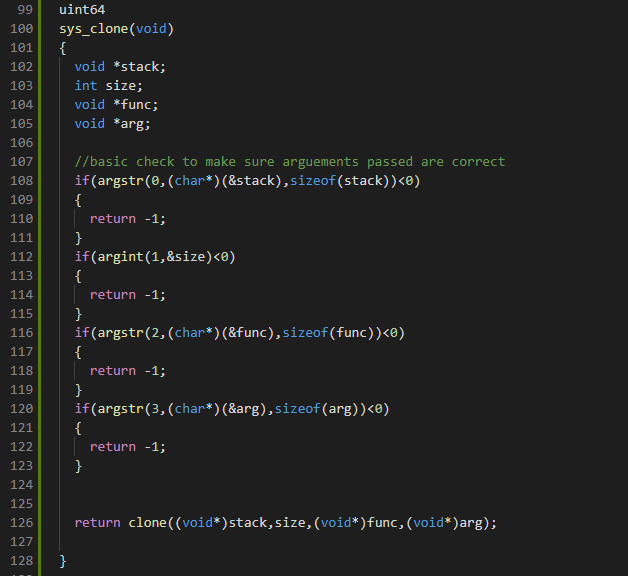
1. kernel/syscall.h

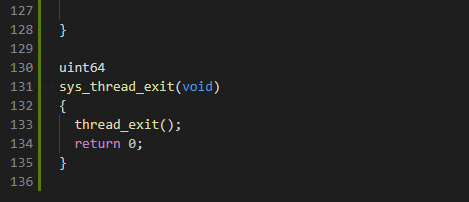
Assign a new number to our system call



1. kernel/sysproc.c

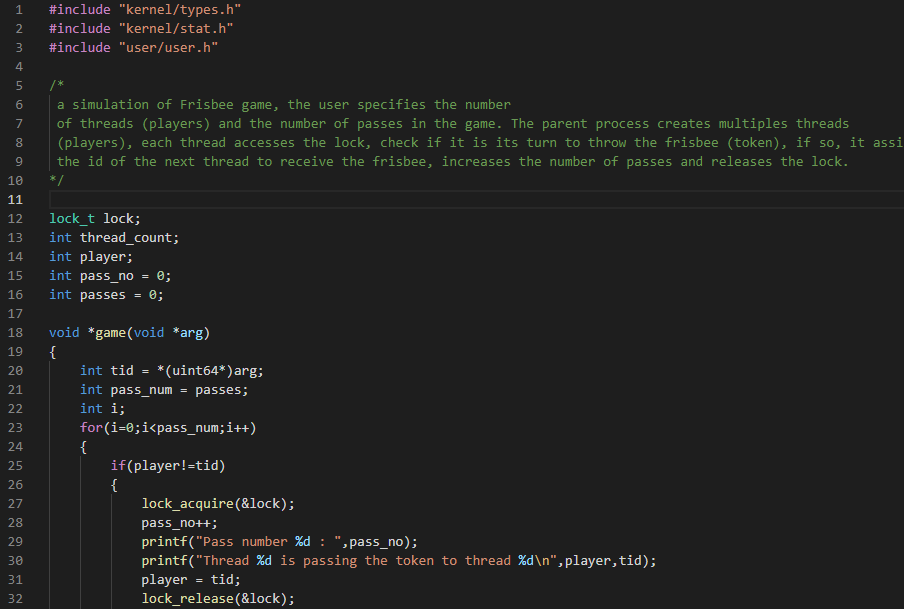
Defined sys\_clone syscall function and passed the argument to clone function for further processing. Silimarly, defined sys\_thread\_exit function and passed the argument to thread\_exit() function.

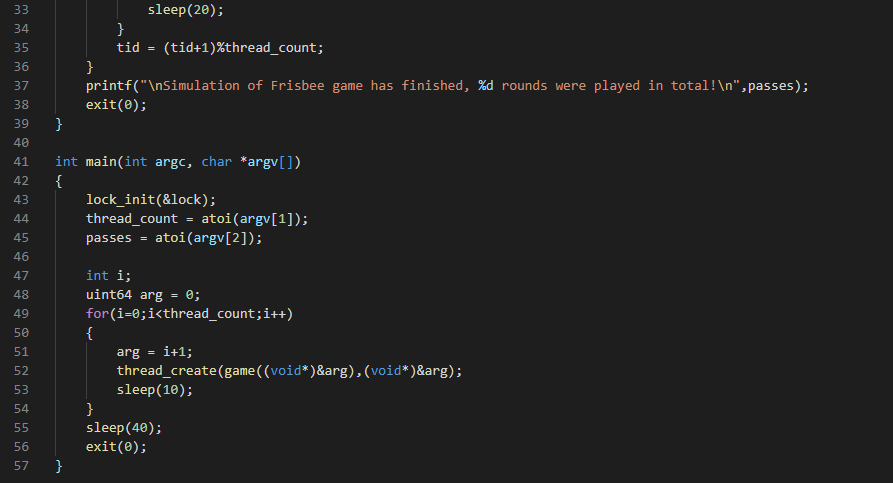




1. kernel/frisbee.c

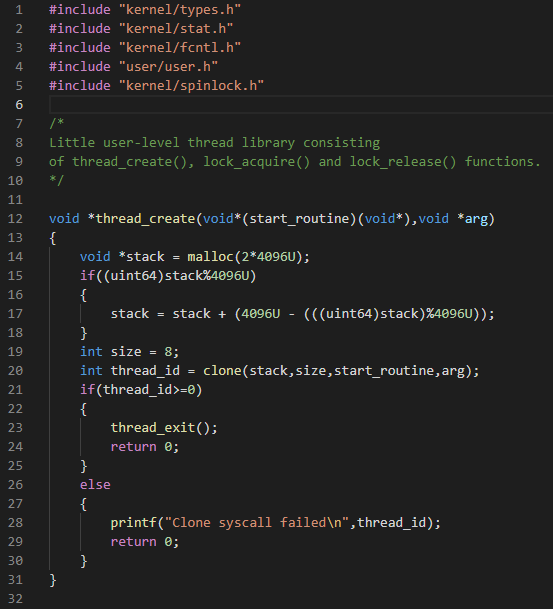
Create user program to test our thread implementation.





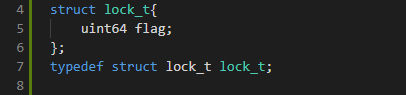
1. user/thread\_lib.c

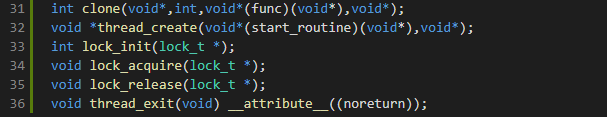
User level thread library with implementation of thread\_create(), lock\_acquire() and lock\_release() function.



1. user/user.h

Added a struct lock\_t for implementing simple user-level spin lock.





1. user/usys.pl

Added entry to update user syscall interface

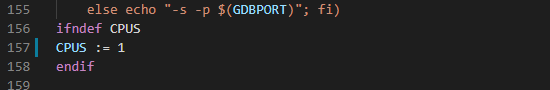


1. Makefile

Added thread\_lib object in user library list. Changed total cpus running to 1.







**Explanation of files**

We have added kernel-level threads support in xv6. First, we implement a clone() system call, which is used to create threads. The clone function creates a thread that shares address space and file descriptors with its parents. The child thread works on a stack different from the parent and created by the parent. Threads need separate trap frames for them, for which we made modifications in the clone function. Since the file descriptors are shared among threads, the exit() and wait() thread needs to be modified. We also modified the wait() function in proc.c. We created another system call and function from exiting the thread called thread\_exit().

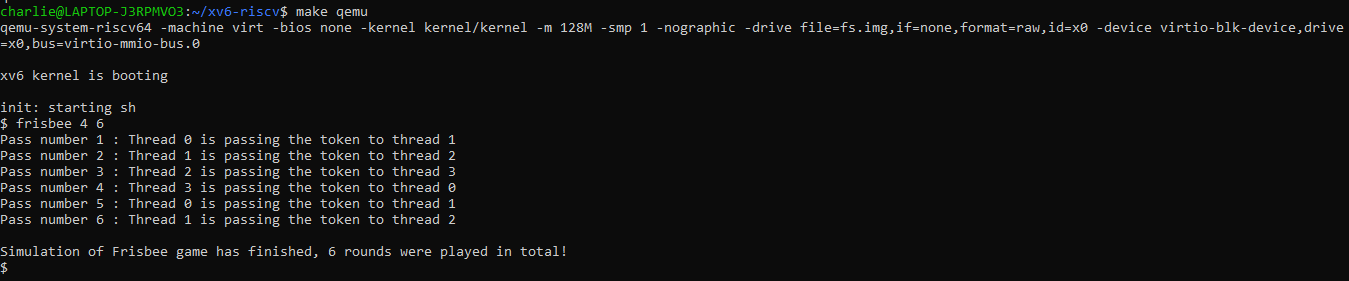
A user-level thread library is created. Functions implemented are thread\_create(), lock\_init(), lock\_aquire() and lock\_release(). The start\_routine uses a clone to call and create the thread. A lock is initialized, acquired, and released using its respective functions. The atomic exchange is implemented using the in-built sync\_test\_and\_set function. Using the frisbee.c program, we demonstrated our work of implementing kernel threads in xv6.

**Output screenshots:**

make clean;

make qemu

$frisbee 4 6



$frisbee 2 6

