**CS202 - Advanced Operating System**

**LAB 3**

* Tejas Milind Deshpande , sid- 862393211
* Satya sri Nandan Paritala , sid - 862395423
* Nunna Lakshmi Saranya , sid- 862394536

**A.Demo Video**

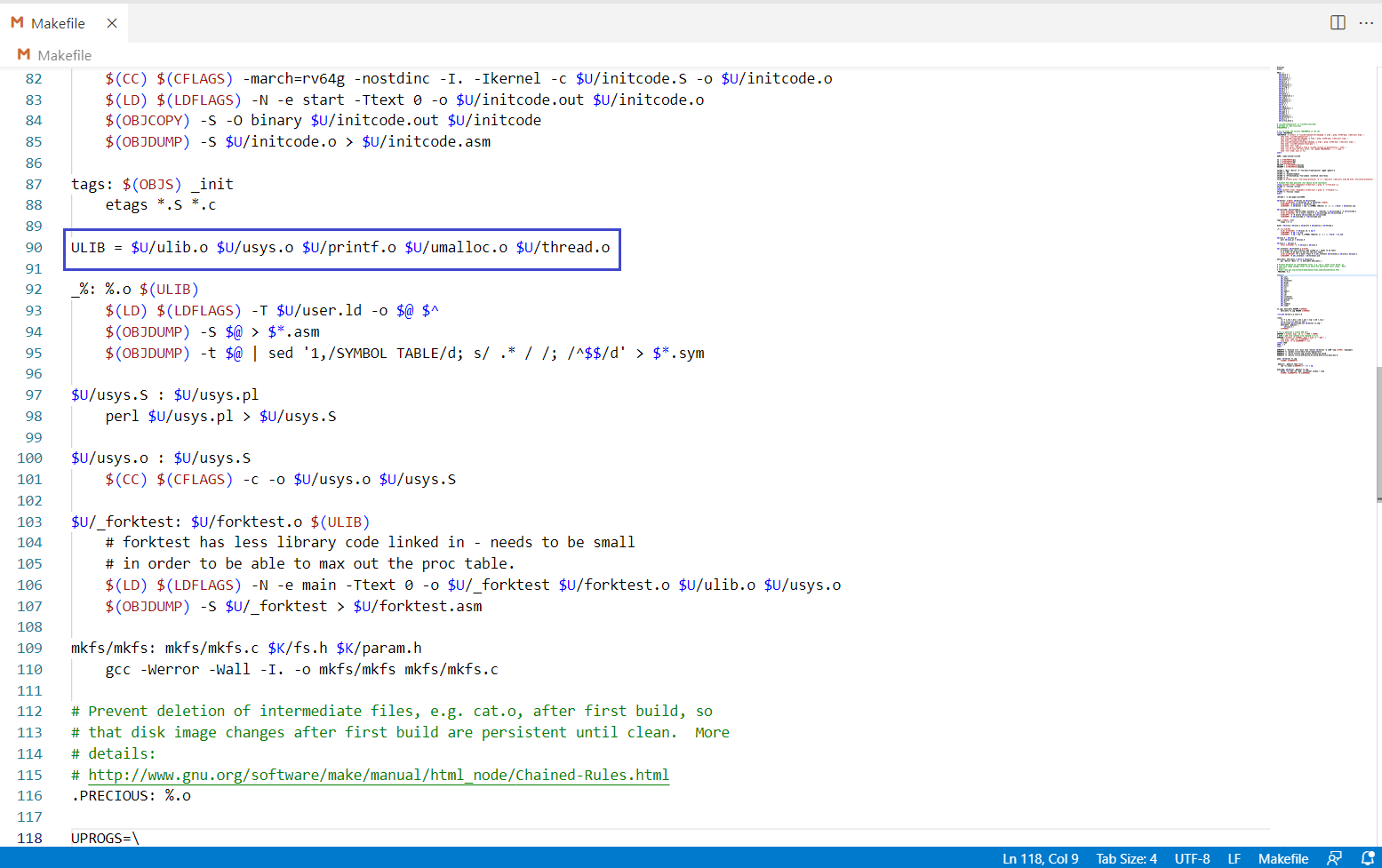
[**https://drive.google.com/file/d/1PMmzMEPbBuLEiun6YTRQdjMcSR2MZumO/view?usp=sharing**](https://drive.google.com/file/d/1PMmzMEPbBuLEiun6YTRQdjMcSR2MZumO/view?usp=sharing)

**B.List of all modified files-**

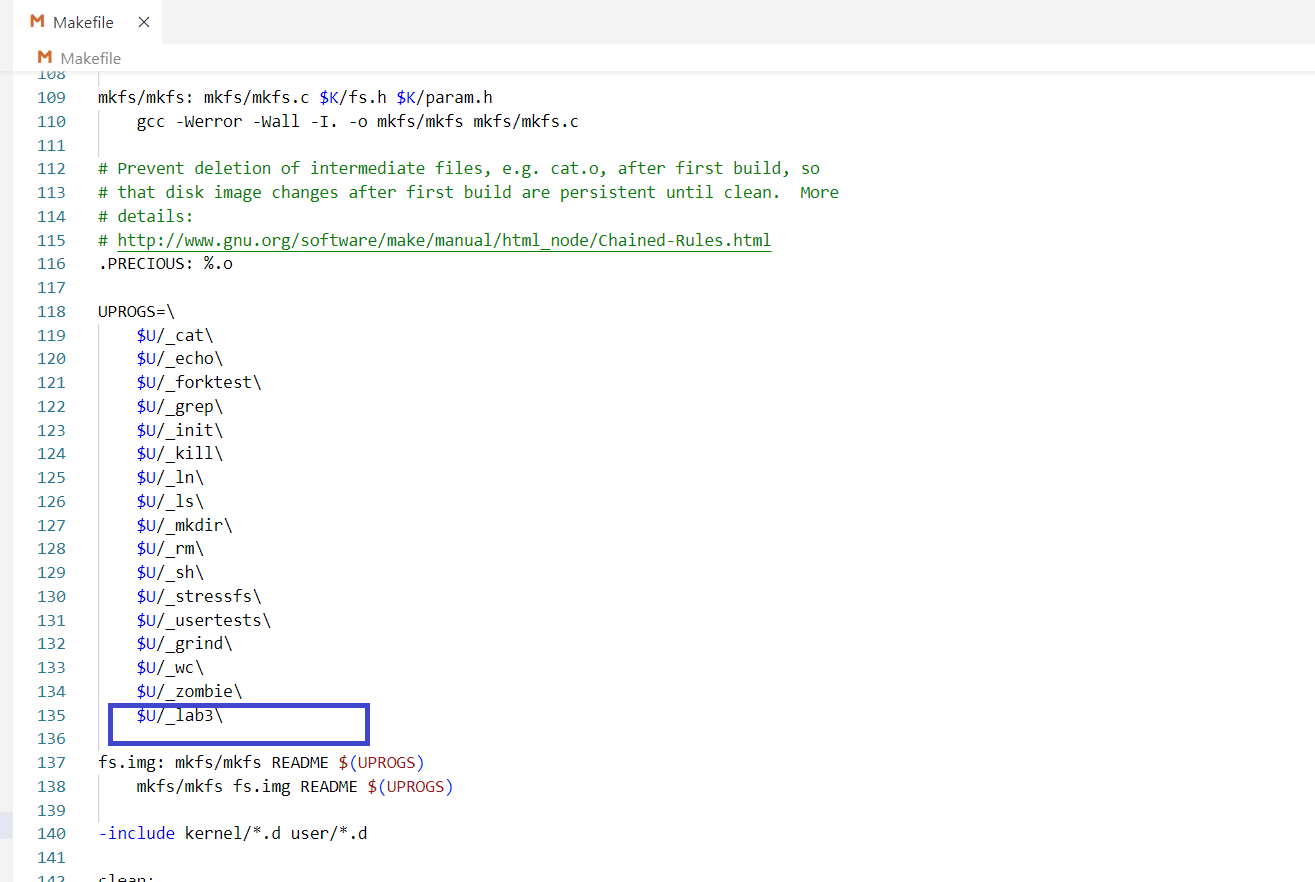
* MAKEFILE
* kernel/defs.h
* kernel/proc.c
* kernel/proc.h
* kernel/syscall.c
* kernel/syscall.h
* kernel/sysproc.c
* user/user.h
* user/usys.pl
* trampoline.S (provided in Lab3 PDF)
* kernel/trap.c
* user/thread.c
* user/thread.h
* New file for testing - user/lab3.c

**C. Changes and screenshots-**

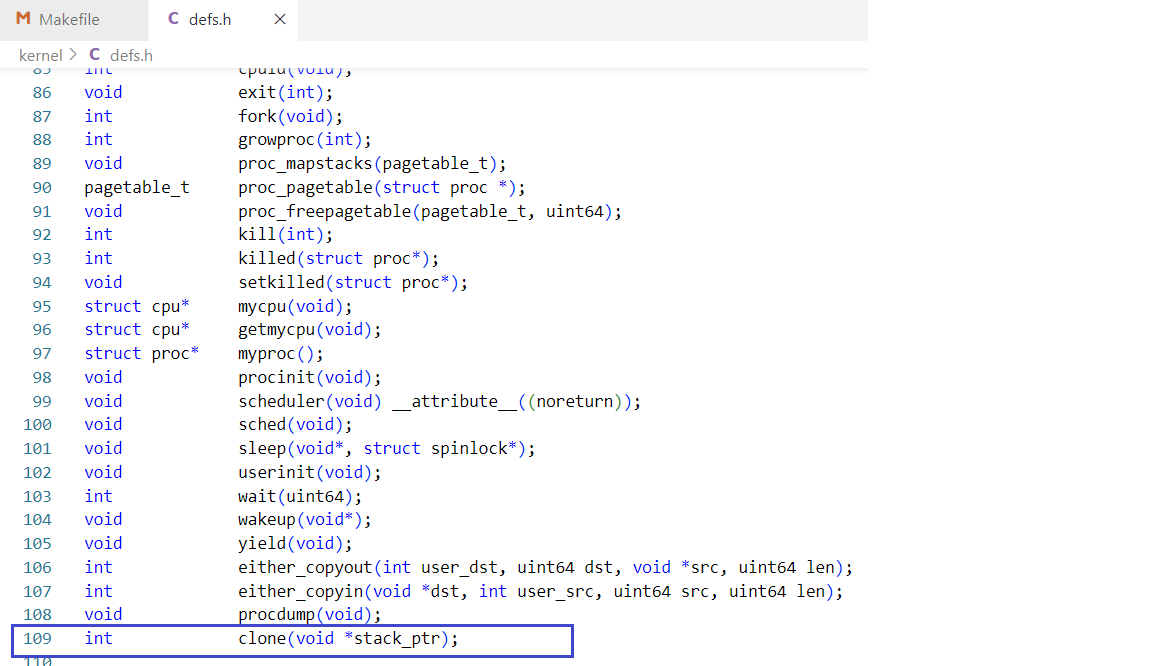
1. **Makefile:**



1. Adding thread library for loading during compilation.

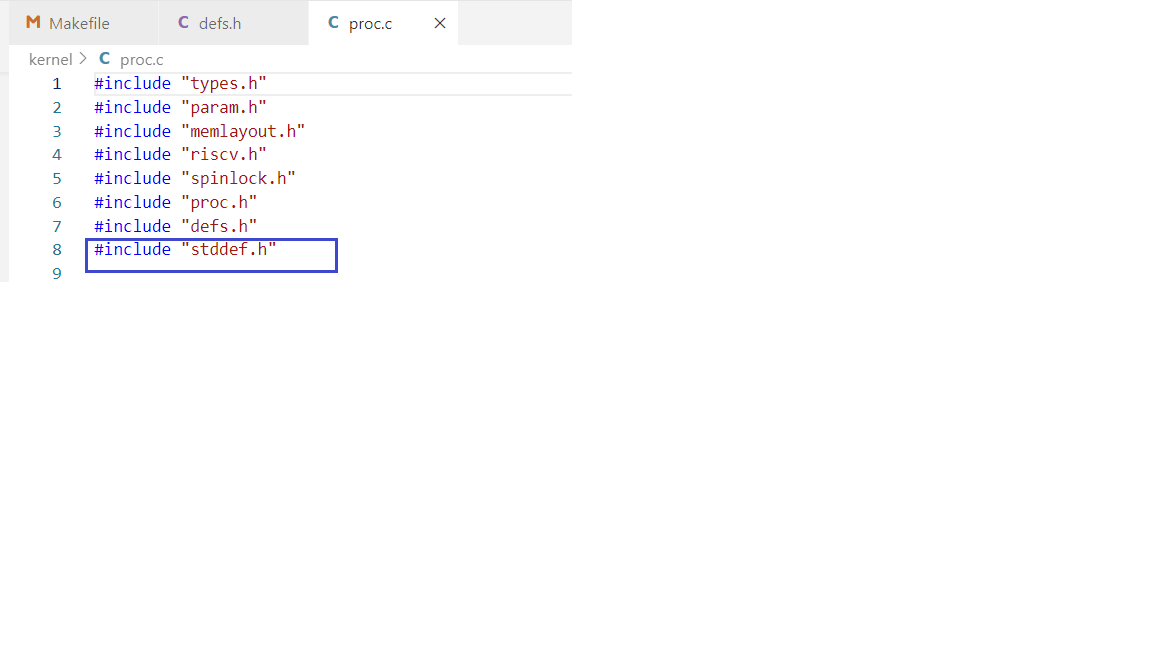


1. Adding lab3 user level program.
2. **kernel/defs.h:**

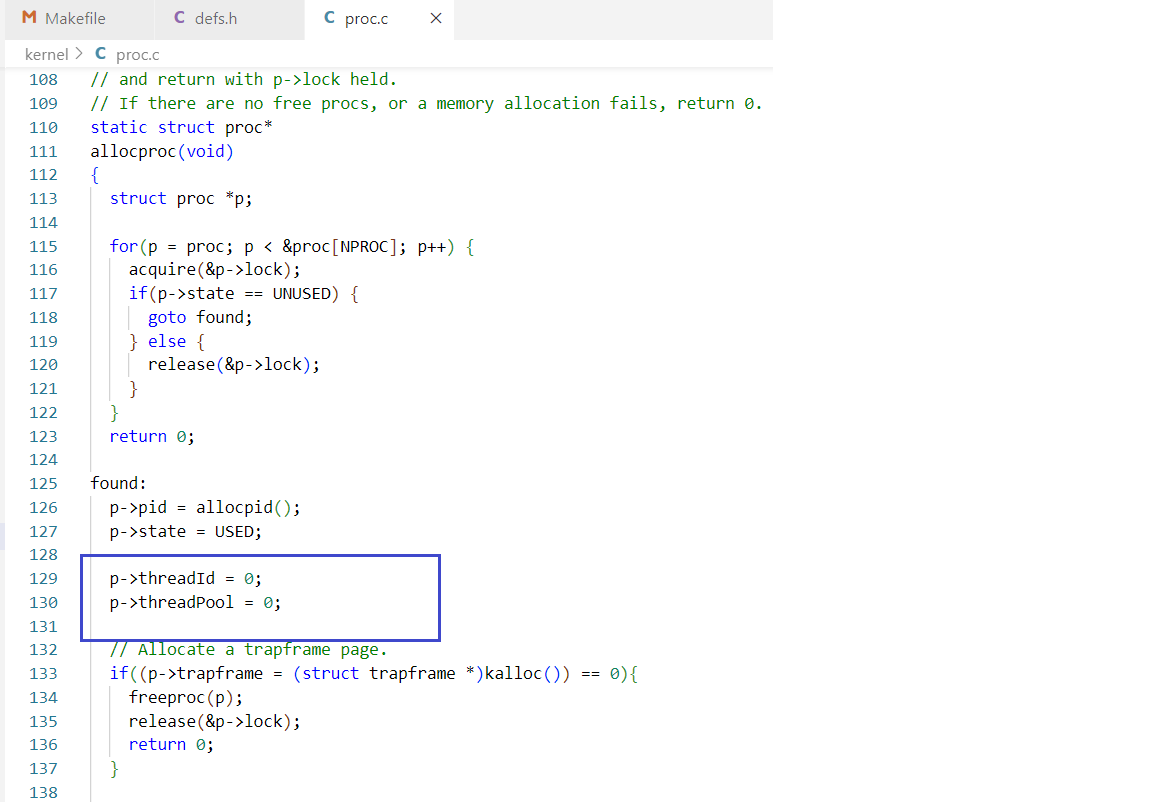
****

Declare clone kernel level call.

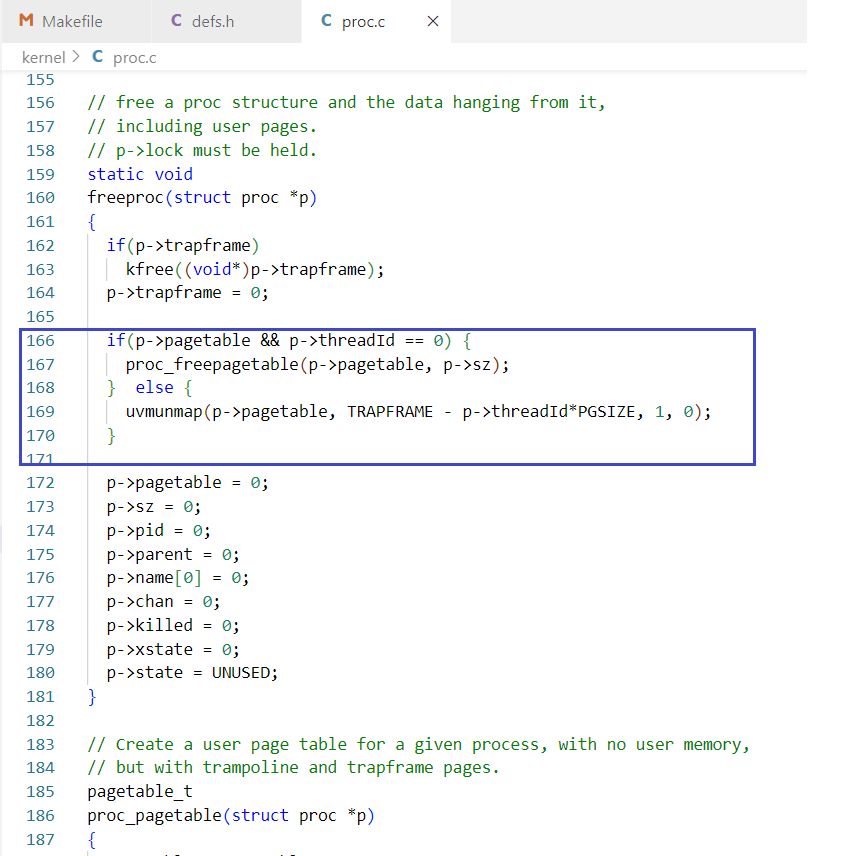
1. **kernel/proc.c:**

****

1. Adding standard definitions library for **NULL** definition.

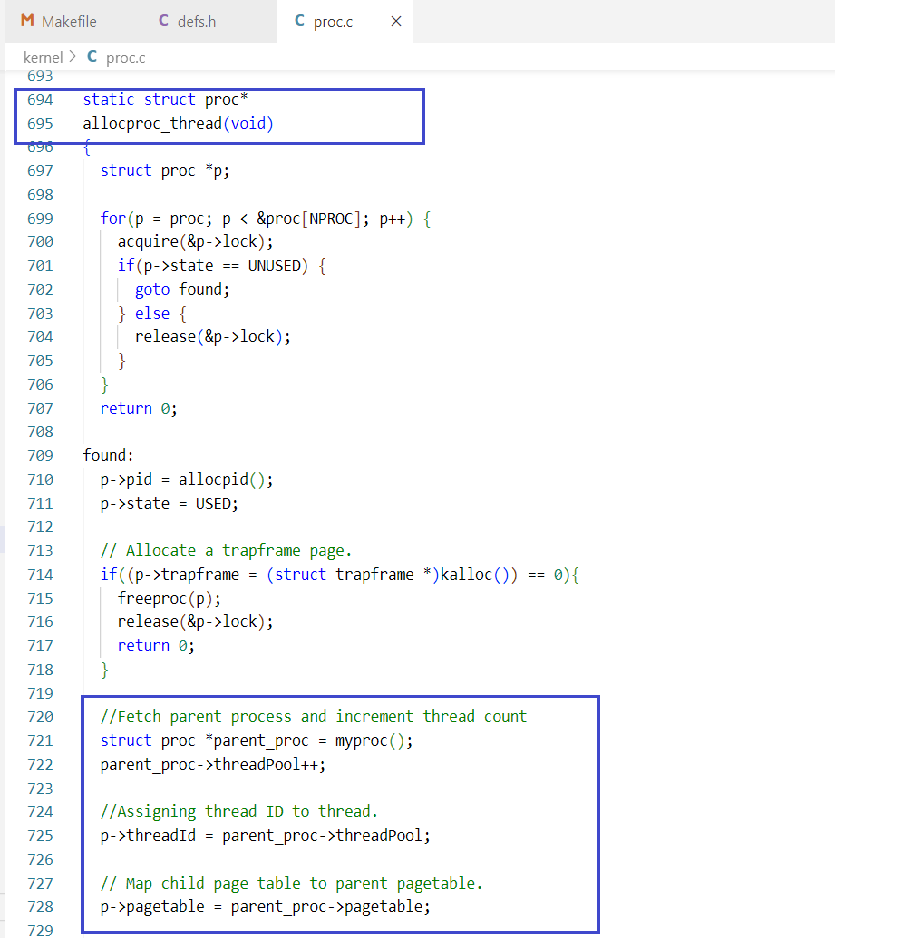


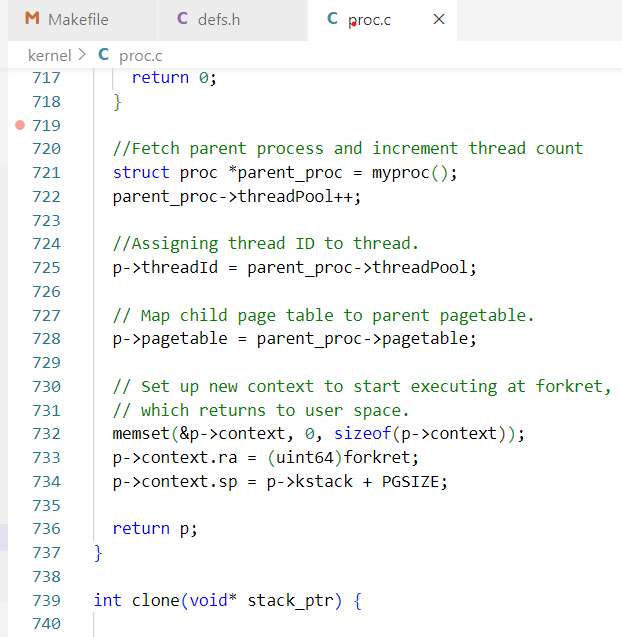
1. Initialize thread ID and total number of threads to zero while creating a process in **allocproc**.



1. If a **parent** process has called **freeproc** then deallocate the page table.

If a child **thread** calls freeproc then deallocate the thread’s trapframe from the parent pagetable.



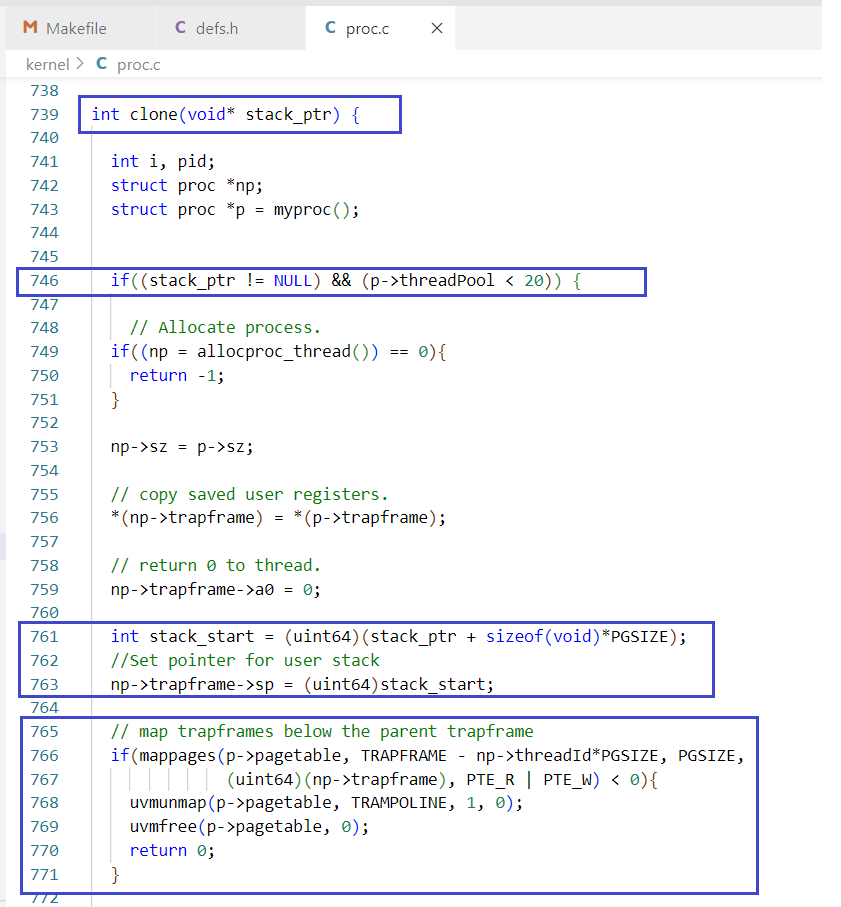


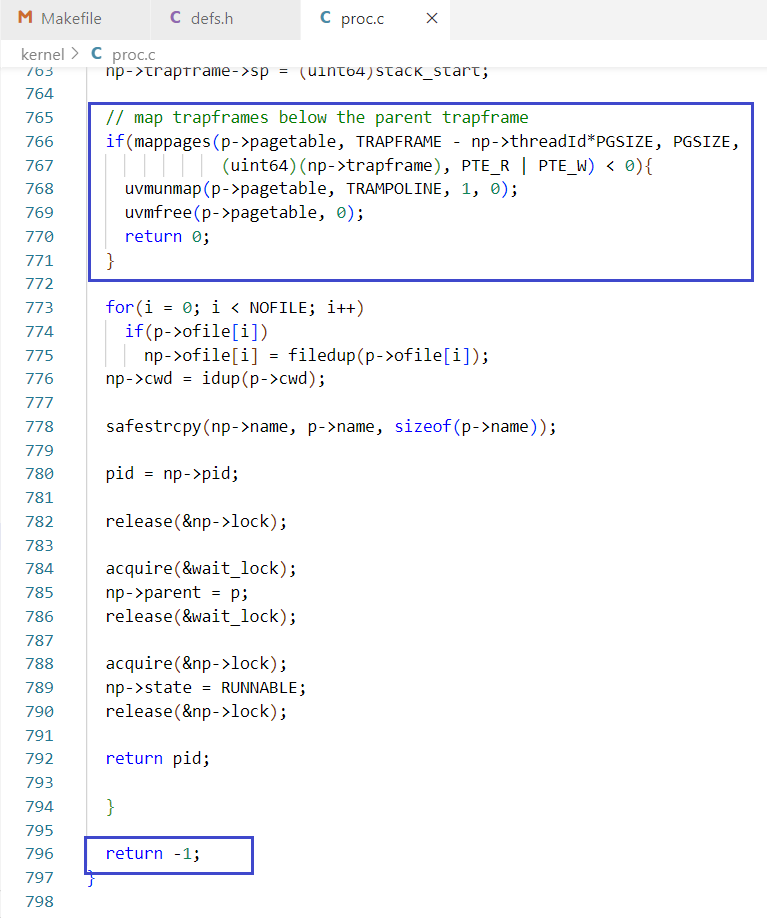
1. The above screenshots are for the new **allocproc\_thread** method.

The changes made to the original allocaproc method for allocating threads instead of processes are highlighted.

Here, we are incrementing the thread count of the parent process which is spawning the thread and assigning thread ID to to the newly created thread.

We are also mapping the child’s pagetable reference to point to the parent pagetable.





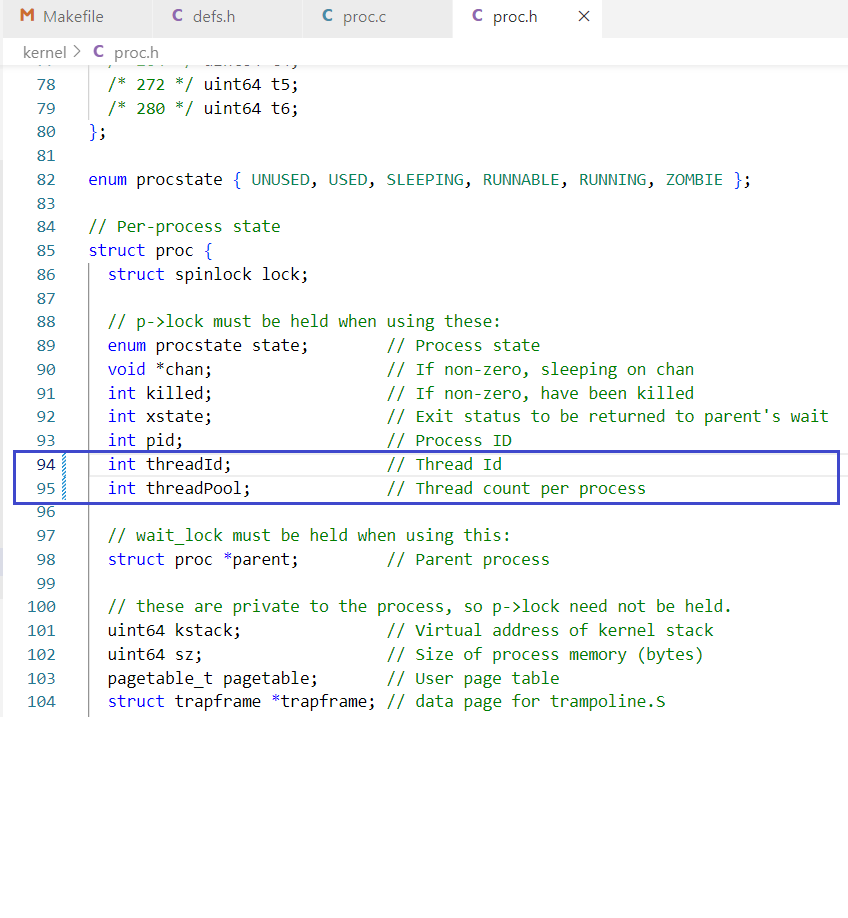
1. The above screenshots show the new **clone()** method, which creates new threads. The changes done to the original **fork()** method are highlighted.

We are receiving the stack pointer parameter and checking performing **Null** check. We are also checking if the maximum thread limit is already reached. If either of these conditions are true, then we exit and return -1 as status for unsuccessful execution.

We are calculating the starting address of the stack and then setting the stack pointer in the new thread’s trapframe to point to this starting address.

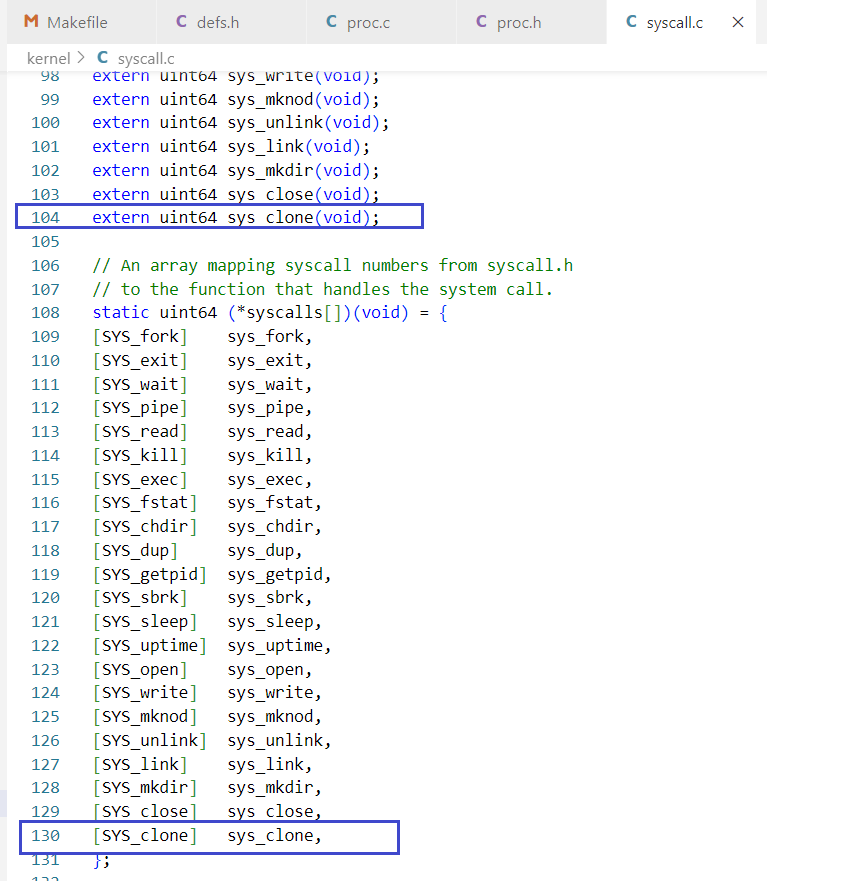
We are also mapping the new thread’s trapframe below the trapframes of the parent and previously created threads in the pagetable of the parent process.

**4. kernel/proc.h:**

****

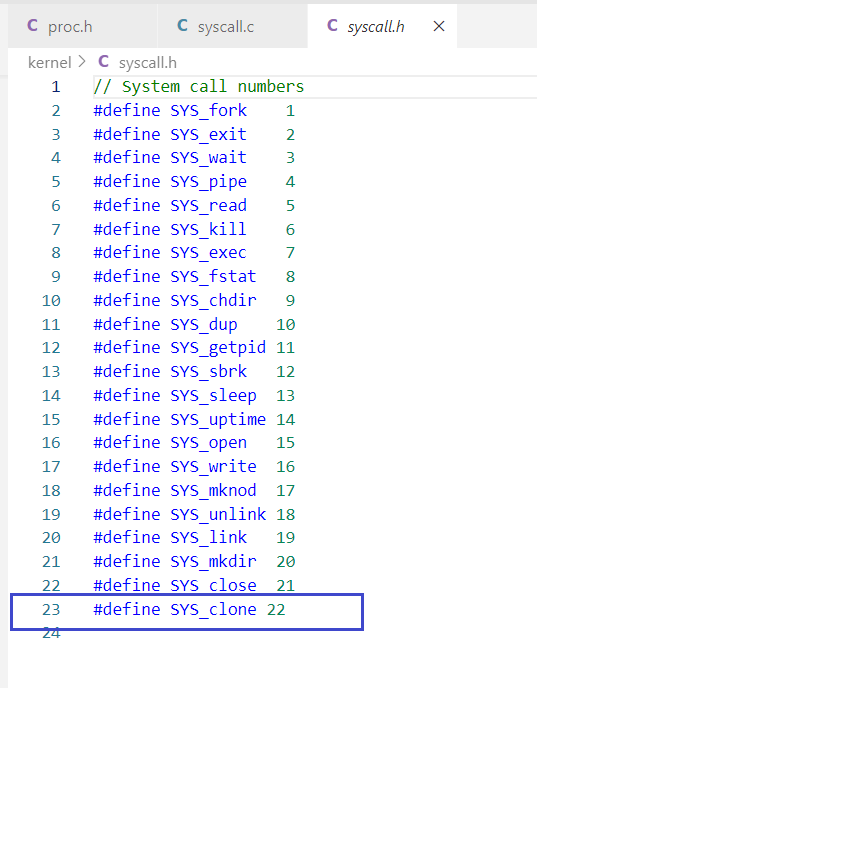
Adding thread ID and thread count in **PCB.**

**5. kernel/syscall.c:**

****

Adding clone system call and exposing it to user level using extern.

**6.** **kernel/syscall.h:**

****

Adding system call entry.

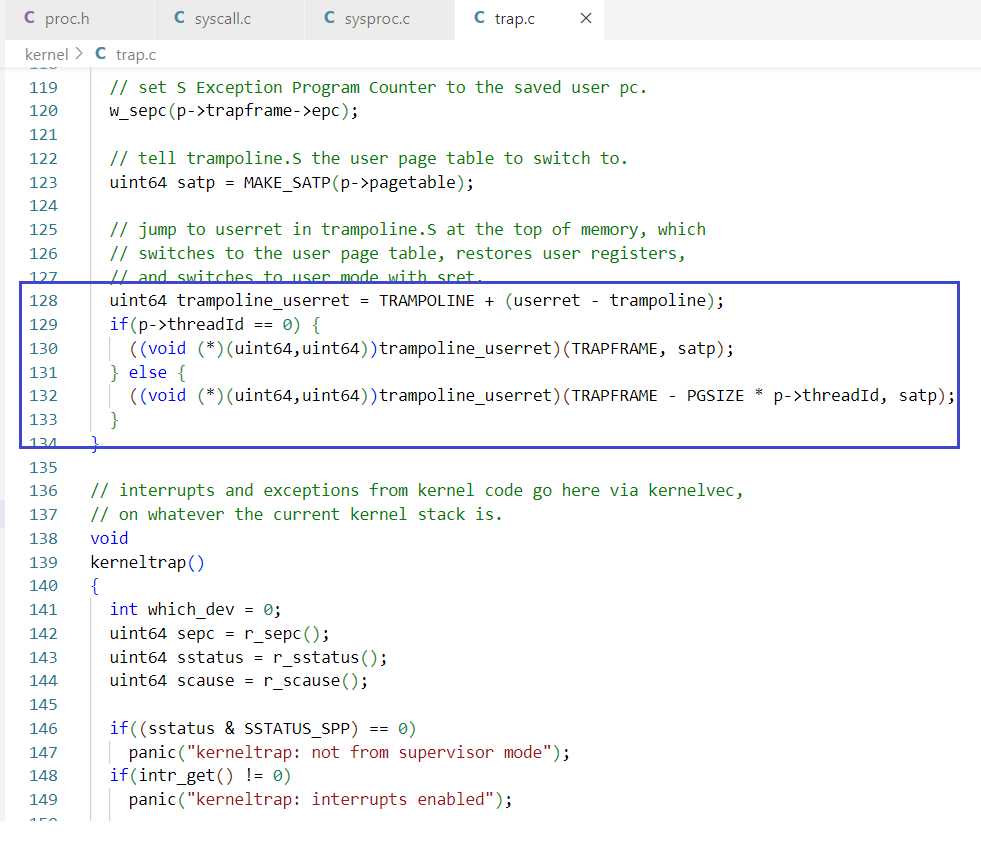
**7.** **kernel/sysproc.c:**



User level system call definition. This function is called from the user level thread library.

It takes the stack pointer and forwards it to the kernel level clone call to create a new thread.

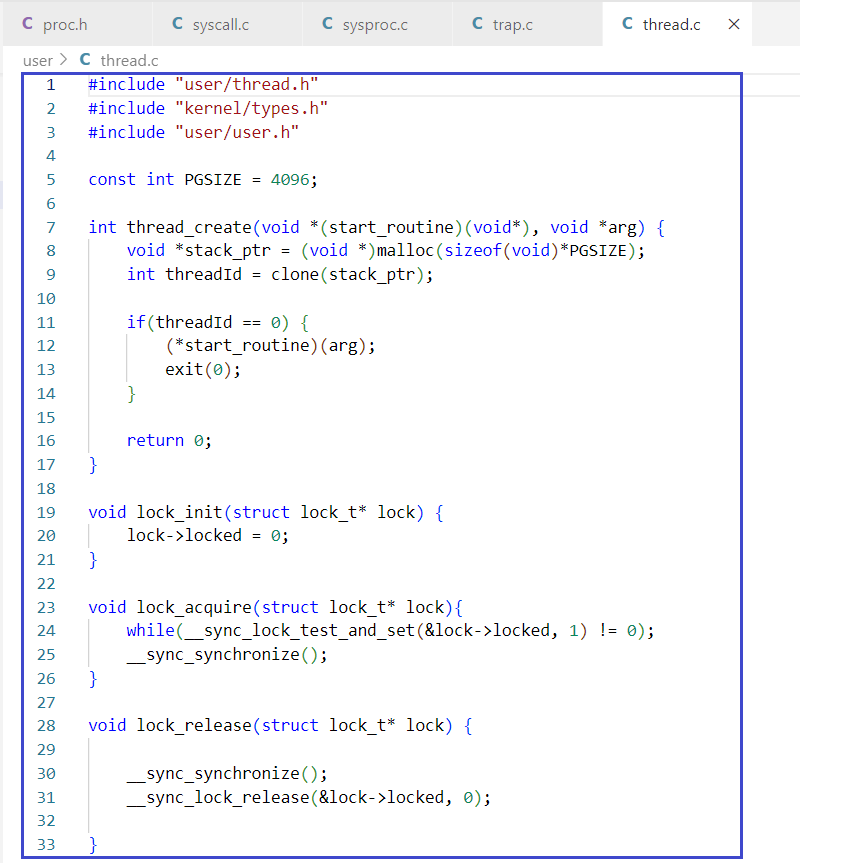
**8.** **kernel/trap.c:**

****

In trap.c, we are are specifying the new trapframe locations for the child threads to the kernel, for

handling the boundary crossing for child threads.

**9.** **user/thread.c:**

****

This file contains the user-level thread library implementations.

In the **thread\_create** function, we are allocating a user-level stack for the new thread and storing the

pointer to this stack. We are passing this stack pointer to the clone system call, which returns the

Thread ID.

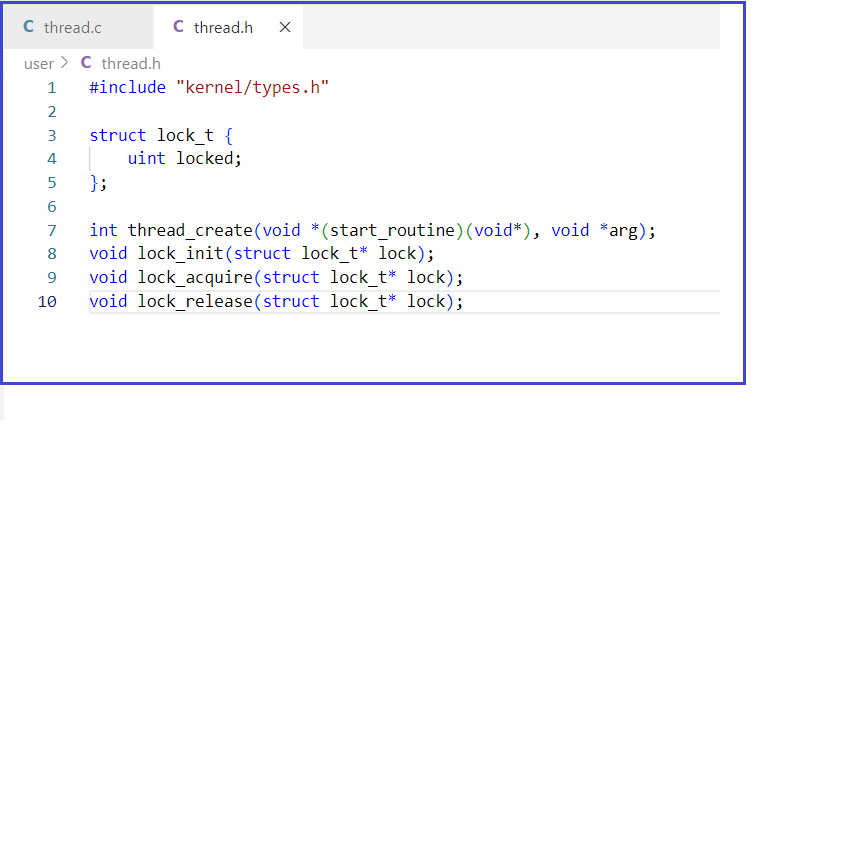
If the thread ID is zero, then we assign the **start\_routine** function to the newly created thread. The

thread exits after completing its execution for the start\_routine.

The **lock\_inint** is used to initialize the lock before starting the start\_routine execution.

The **lock\_acquire** and **lock\_release** functions are used by each thread for its turn to pass the token.

**10. user/thread.h**

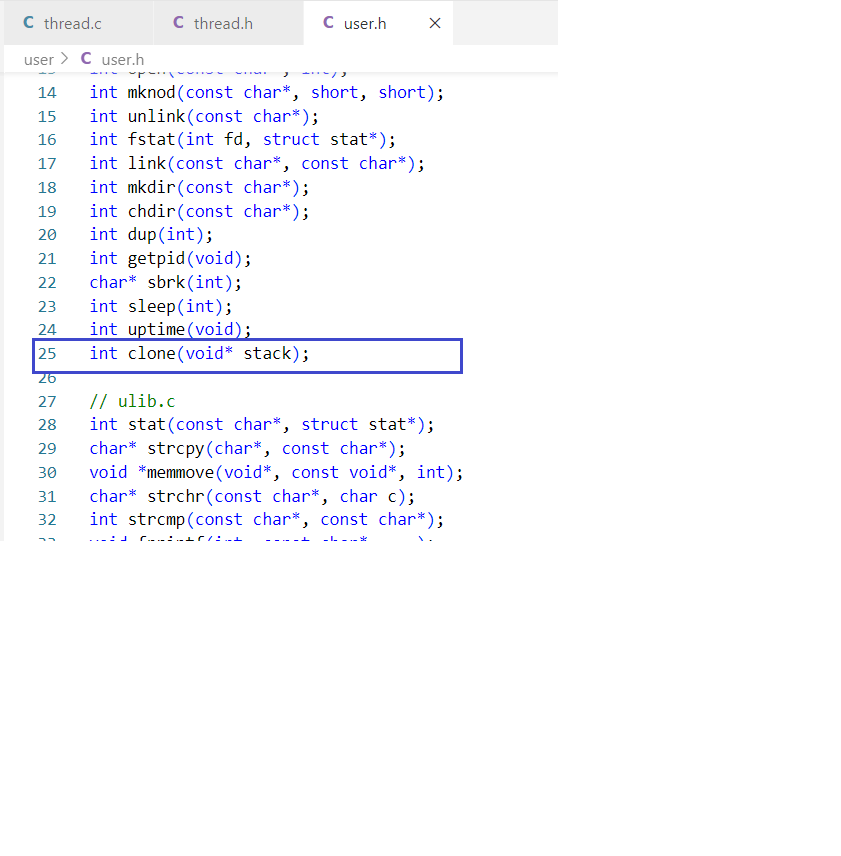
****

This is the library interface for the user level thread library.

It contains the lock\_t structure, which is passed by each thread to acquire and release the lock during

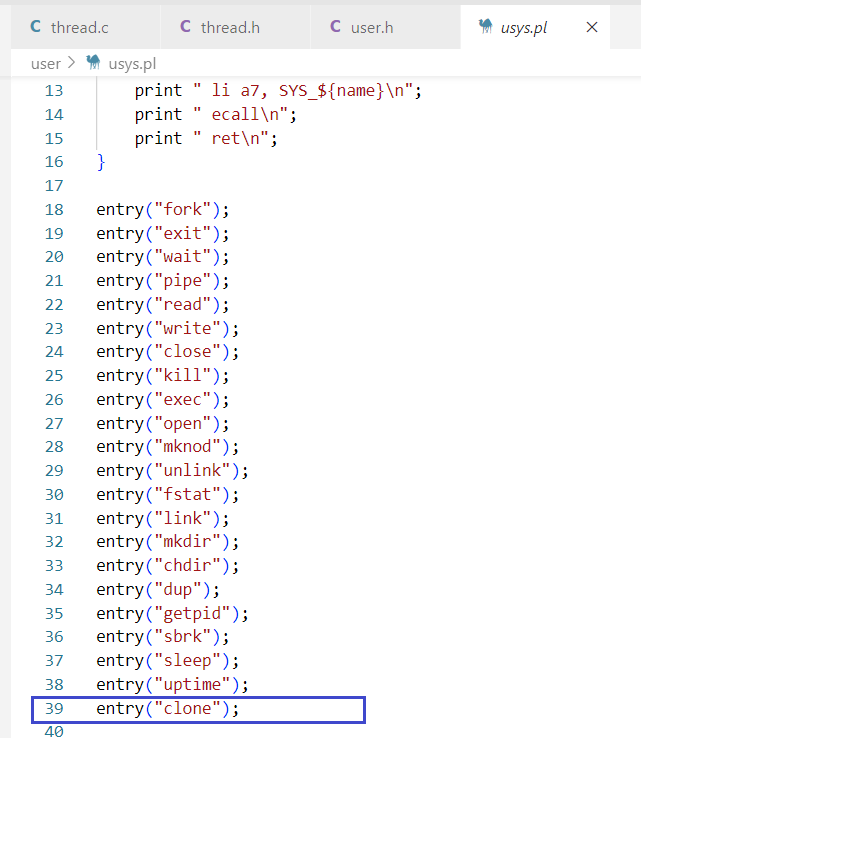
its turn to pass the token.

**11. user/user.h:**

****

Exposing the clone system call in user space sys\_call interface.

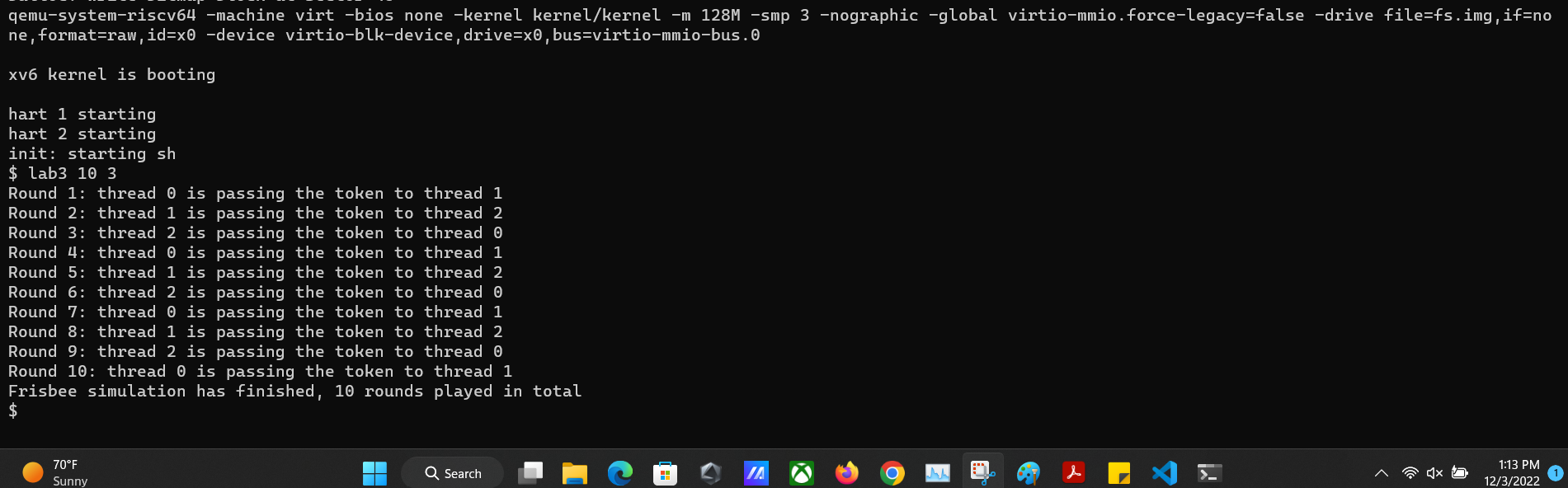
**12. user/usys.pl**

****

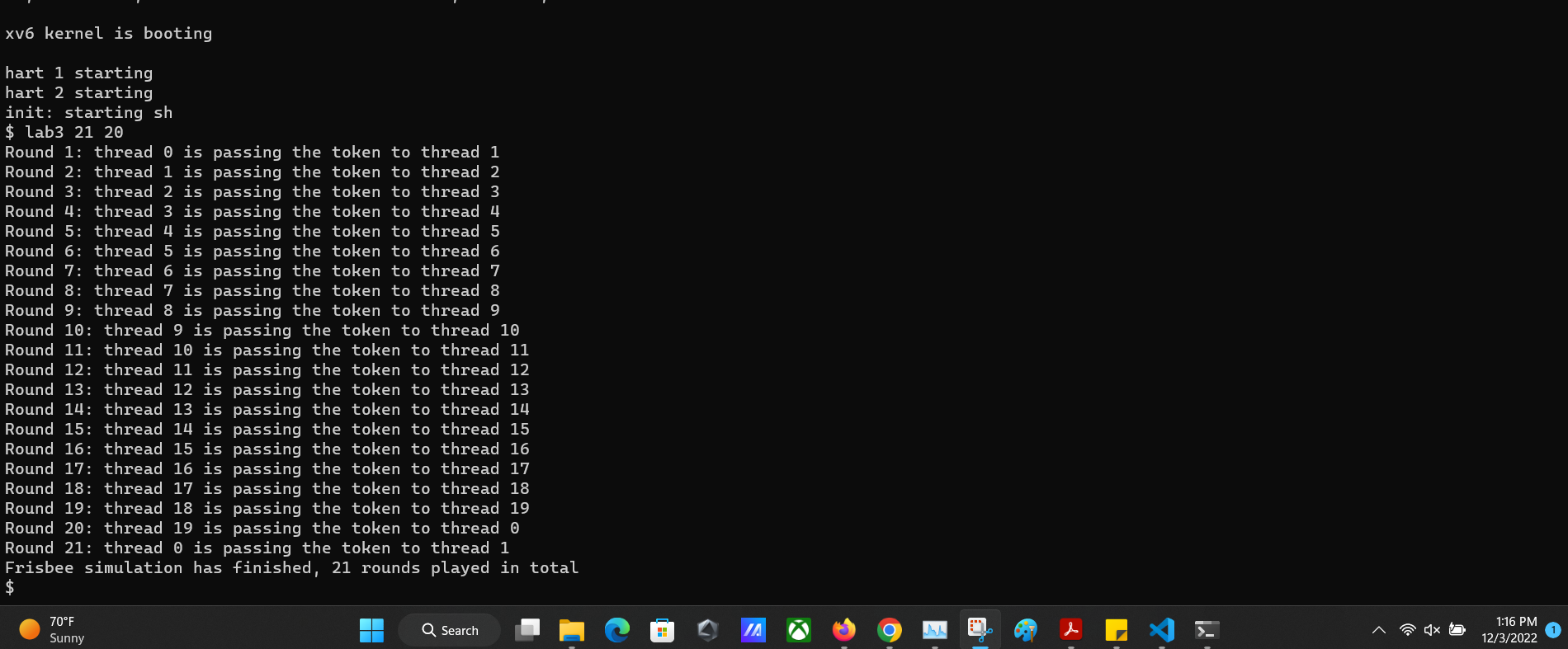
Exposing the clone system call in user space sys\_call interface.

**D. Output:**

1. Result for **lab3 10 3**:



1. Result for **lab3 21 20**:



**E . Contributions -**

| **Tejas Milind Deshpande** | Implemented the Clone system call  Implemented the user-level Thread library  Worked on creating the report and the video  Gave review comments to improve code for peers.  Created the git repository for collaboration.  Integrated the code with peers  Learnt about XV6 code while debugging Kernel panics |
| --- | --- |
| **Satya Sri Nandan Paritala** | Worked on the thread library implementations  Worked on creating the report and video  Helped in reviewing the code  Learnt about the debugging techniques with XV6 |
| **Nunna Lakshmi Saranya** | Implemented the clone system call  Worked on the report and video  Learnt about the thread library implementation  Studied about Xv6 source code |