LAB #2: xv6 Threads

Group Members:

- Yash Aggarwal (yagga004)
- Nityash Gautam (ngaut006)
- Parth Bhatt (pbhat029)

Demonstration Link:

https://drive.google.com/file/d/1VILYF4fSoqz8npYC9SPanR8kn89_v8kU/view

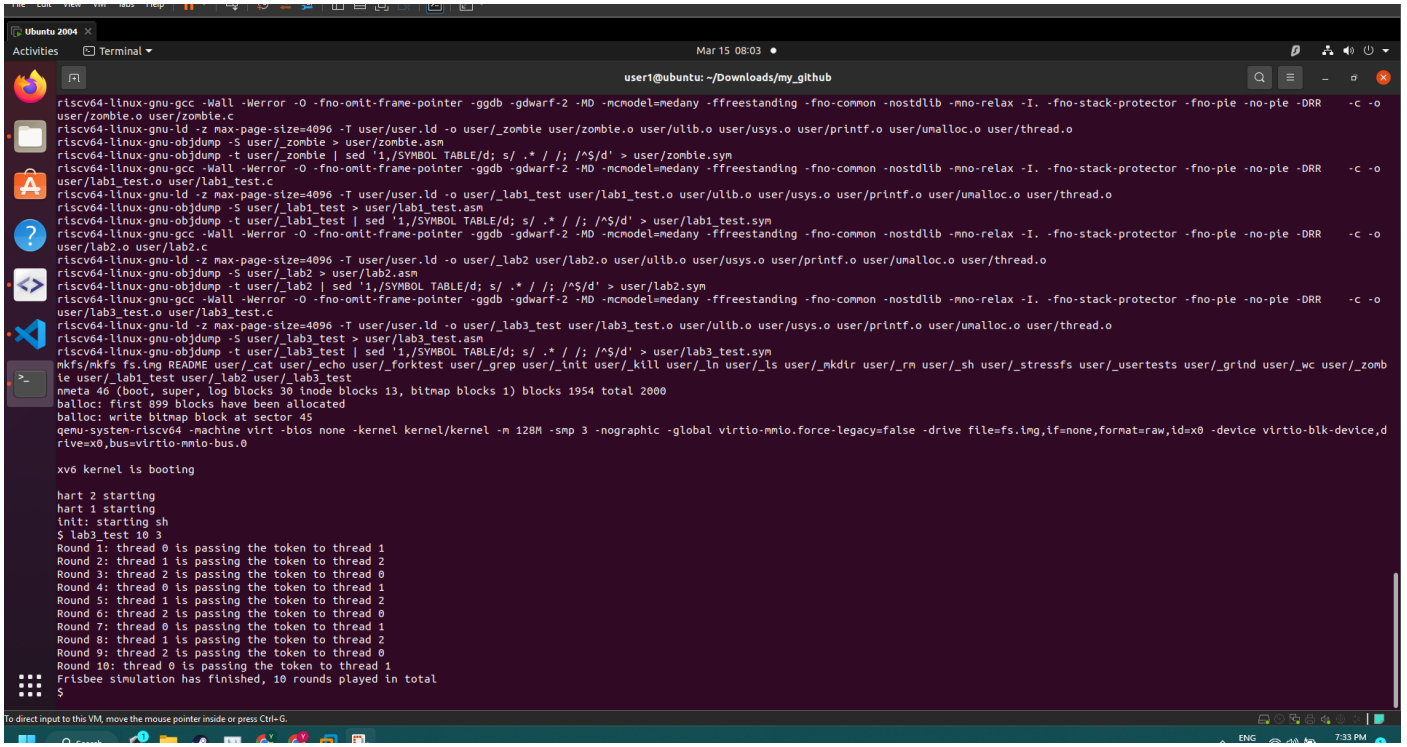
PART 1: Clone() System Call

1. Add the system call in syscall.h
2. Add the mapped function in syscall.c
3. Define the mapped function in syscall.c
4. Implement the syscall function in sysproc.c
5. Add the implementation of clone function in proc.c
6. Add a thread_id variable in proc.h
7. Make changes for the freeproc and uvmunmap() function.
8. Implement the allocproc() function with changes
9. For clone() remember:
    a. Stack null check
    b. Checking of page table
    c. Starting address of child's user stack

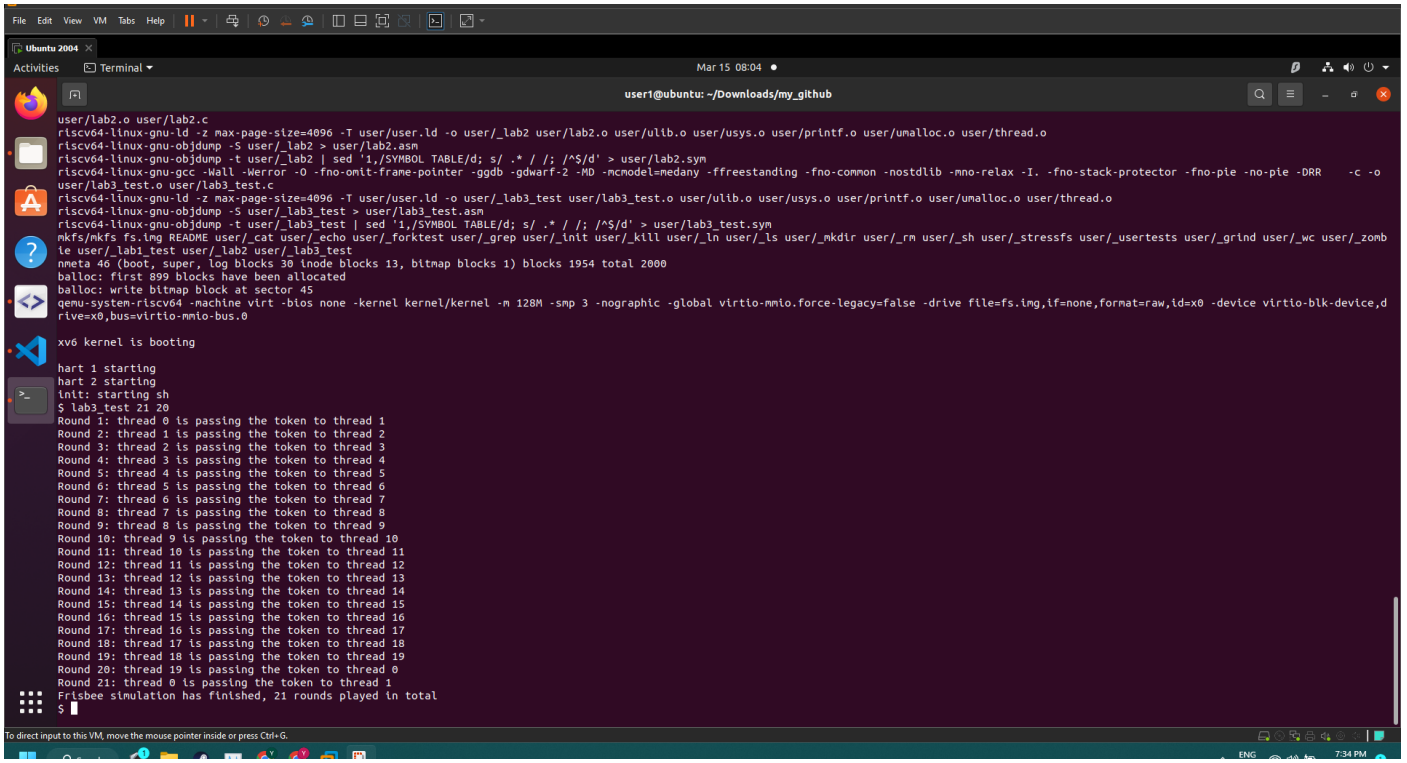
PART 2: User Level Thread Library

1. Create the files for user thread library - thread.c and thread.h
2. Define the schema or structure for the thread.
3. Create empty functions for thread create lock_init, acquire, release
4. Initialize lock as 0.
5. Implement the functions.
6. Add the logic for lock acquire and release sync.
7. Add code for clone
8. Test the implementation

CONCLUSION:



Demo for **lab3_test 10 3**



Demo for **lab3_test 21 20**

CONTRIBUTIONS

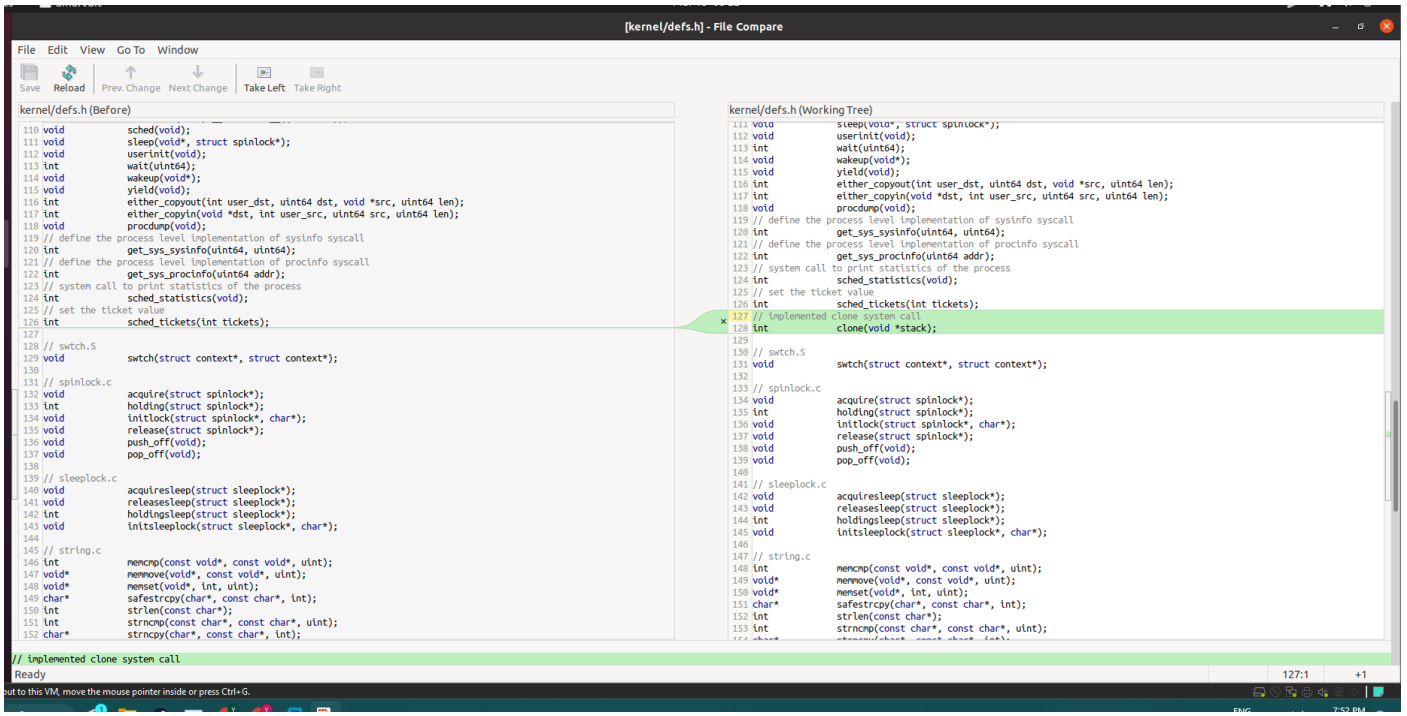
All the members discussed the implementation, strategy and changes before implementation. Also all members helped each other in case there was an issue in implementation.

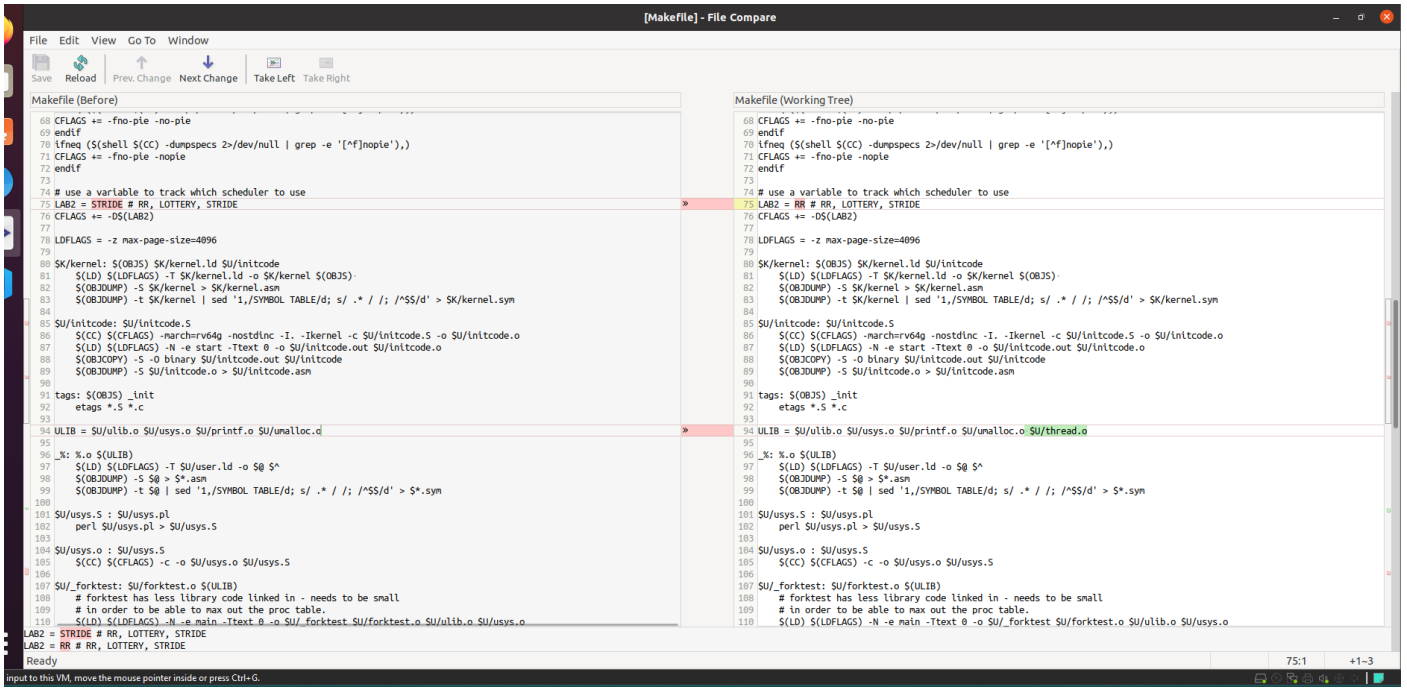
Yash worked on part 1 of the lab and implemented the system call structure. Yash also created the basic schema for the user level thread library in part 2.

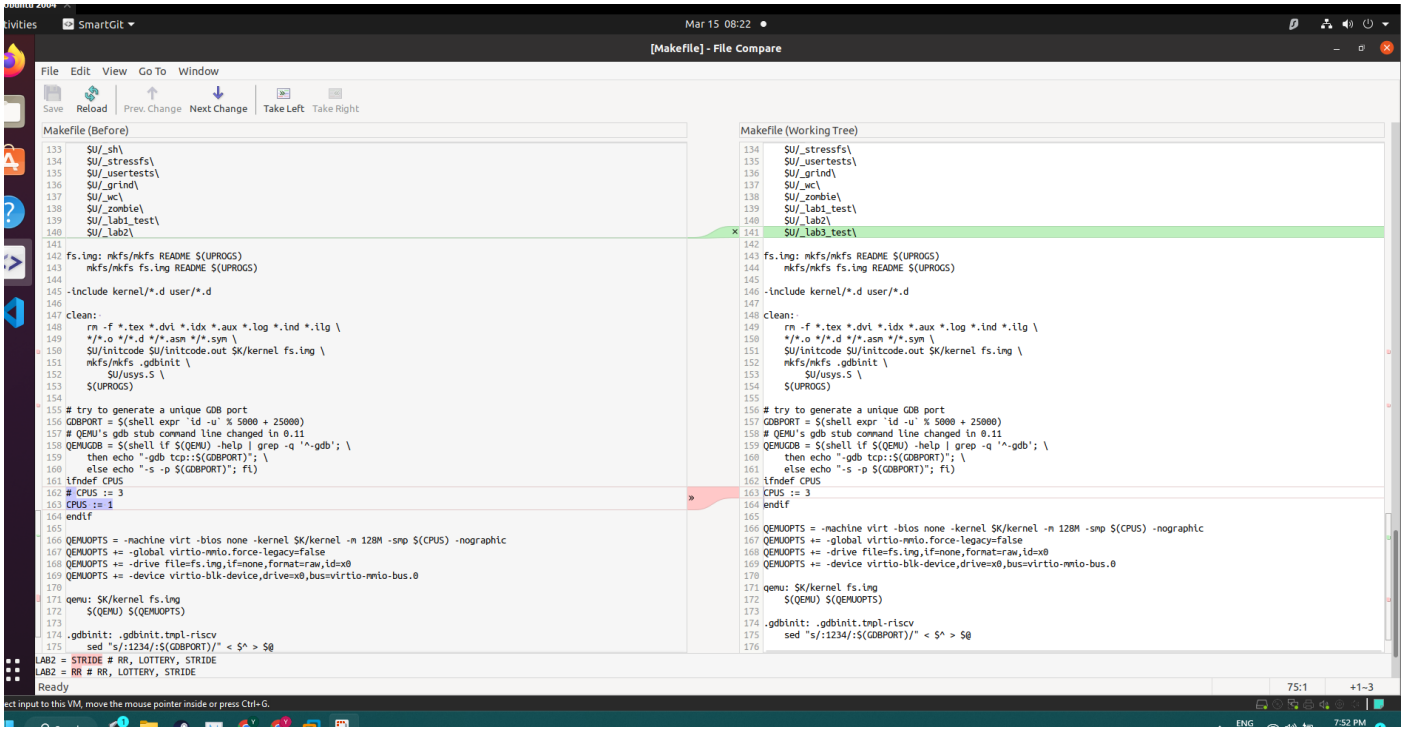
Nityash implemented and added the code for clone in the defined schemas.

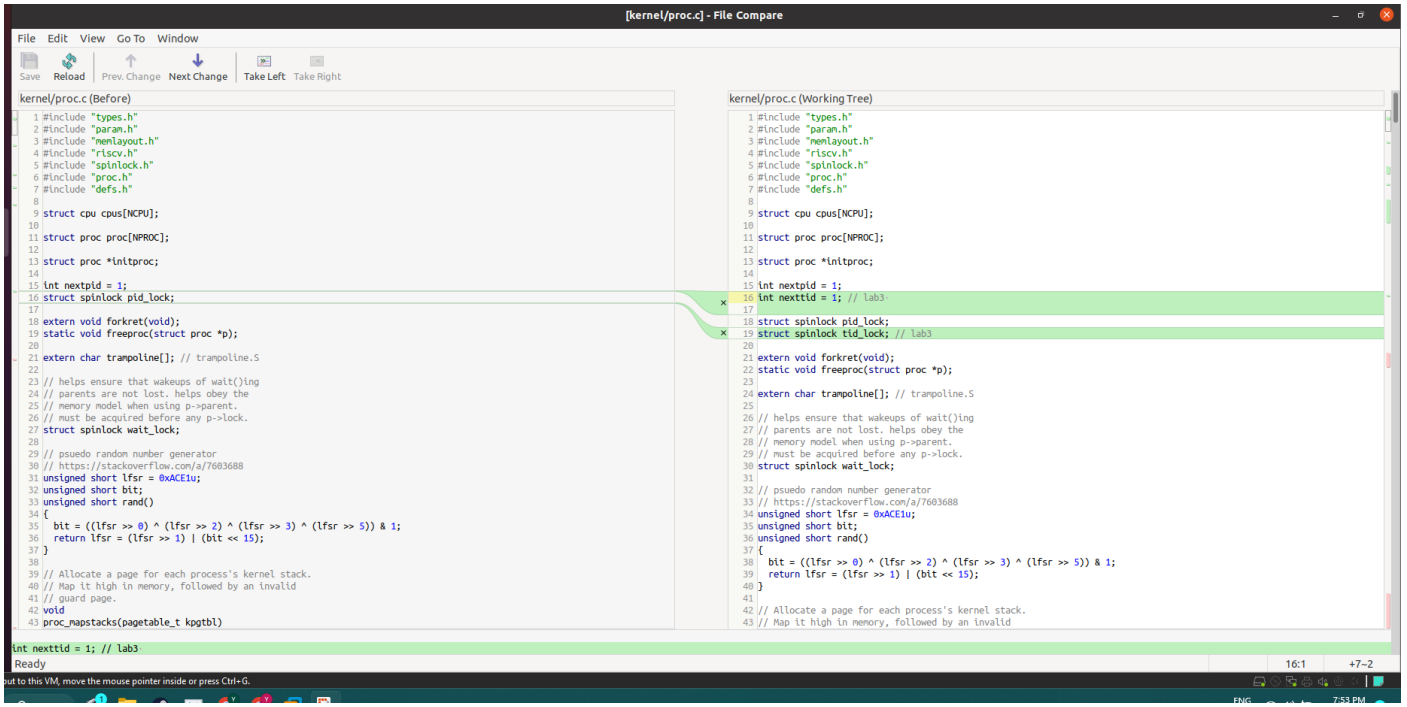
Parth implemented and added the code for user level threads in part 2.

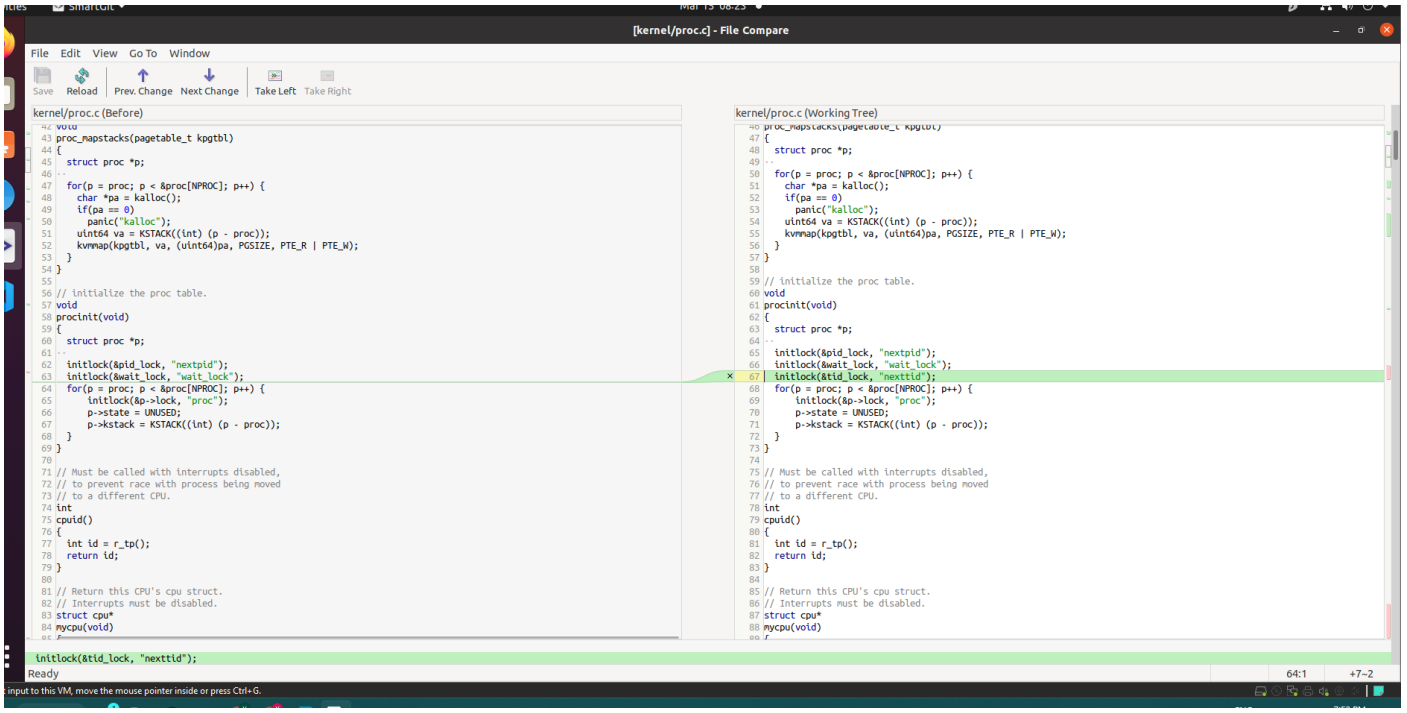
CODE CHANGES

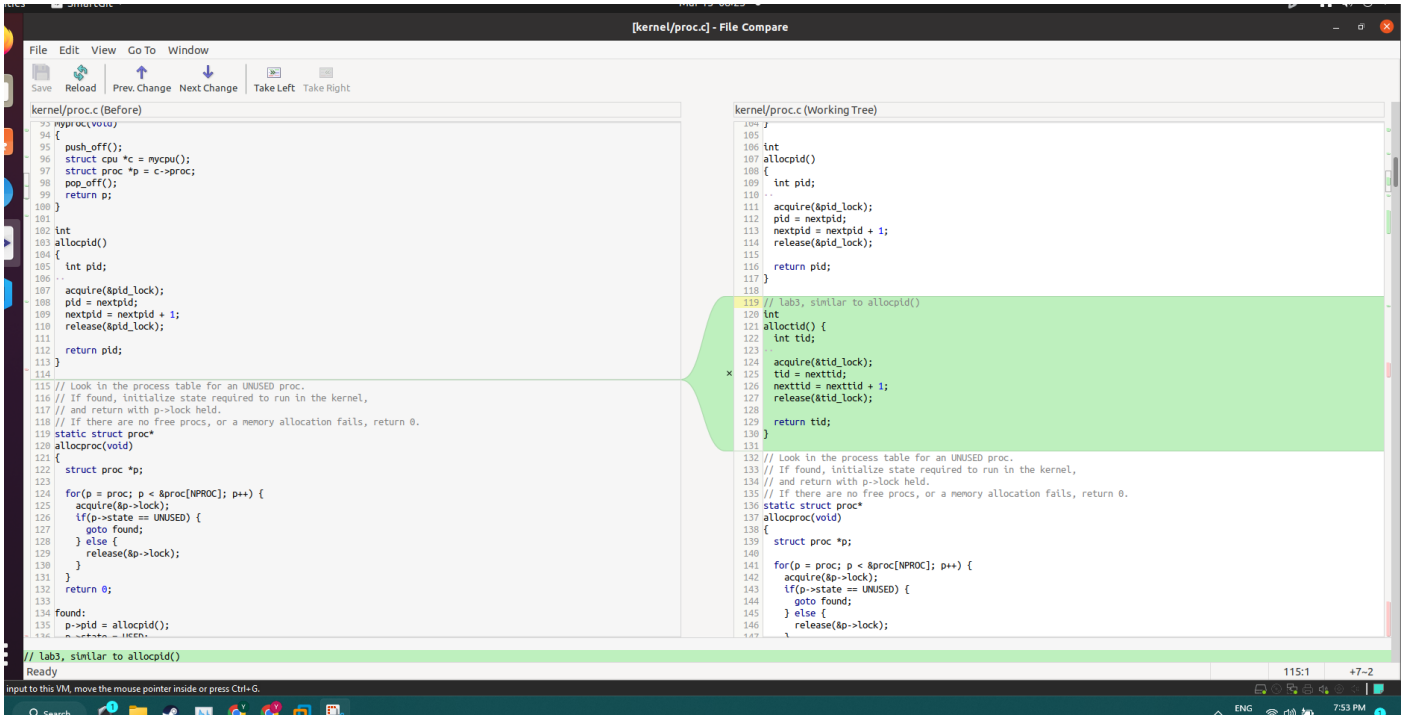


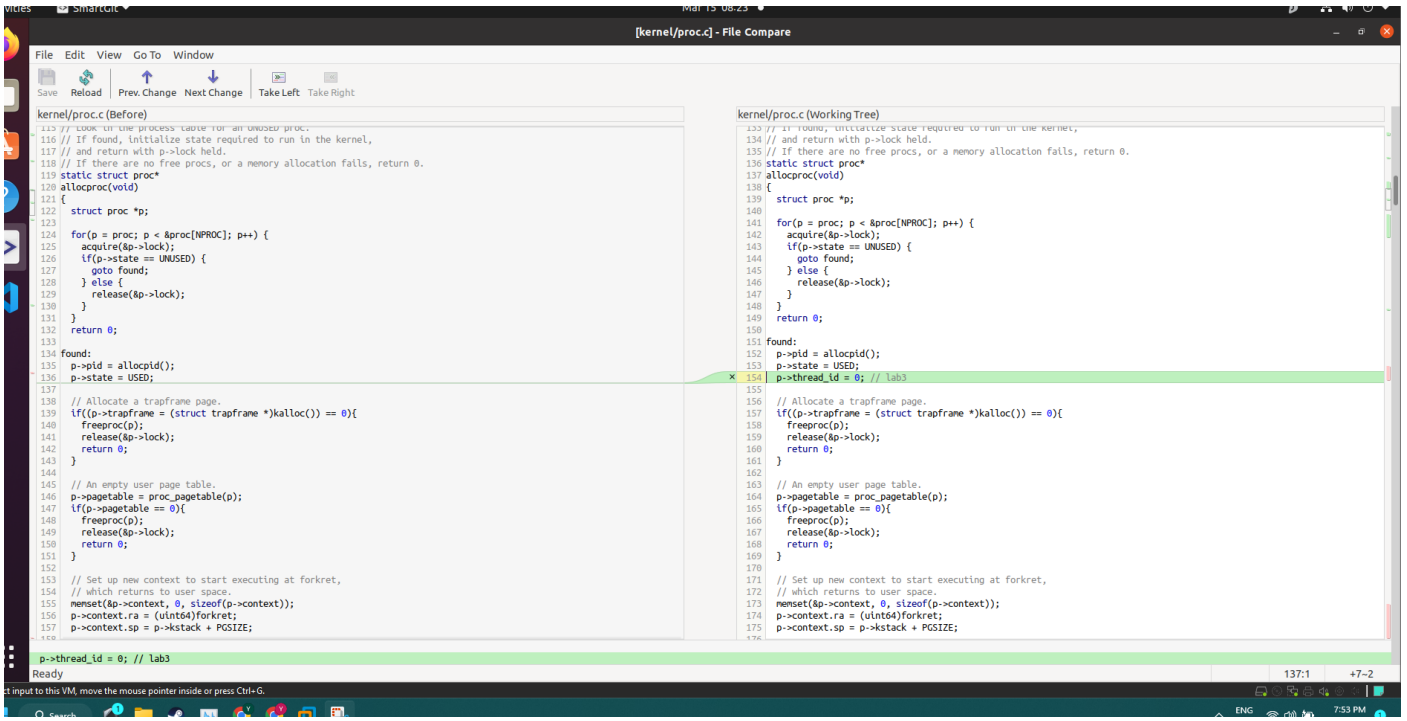


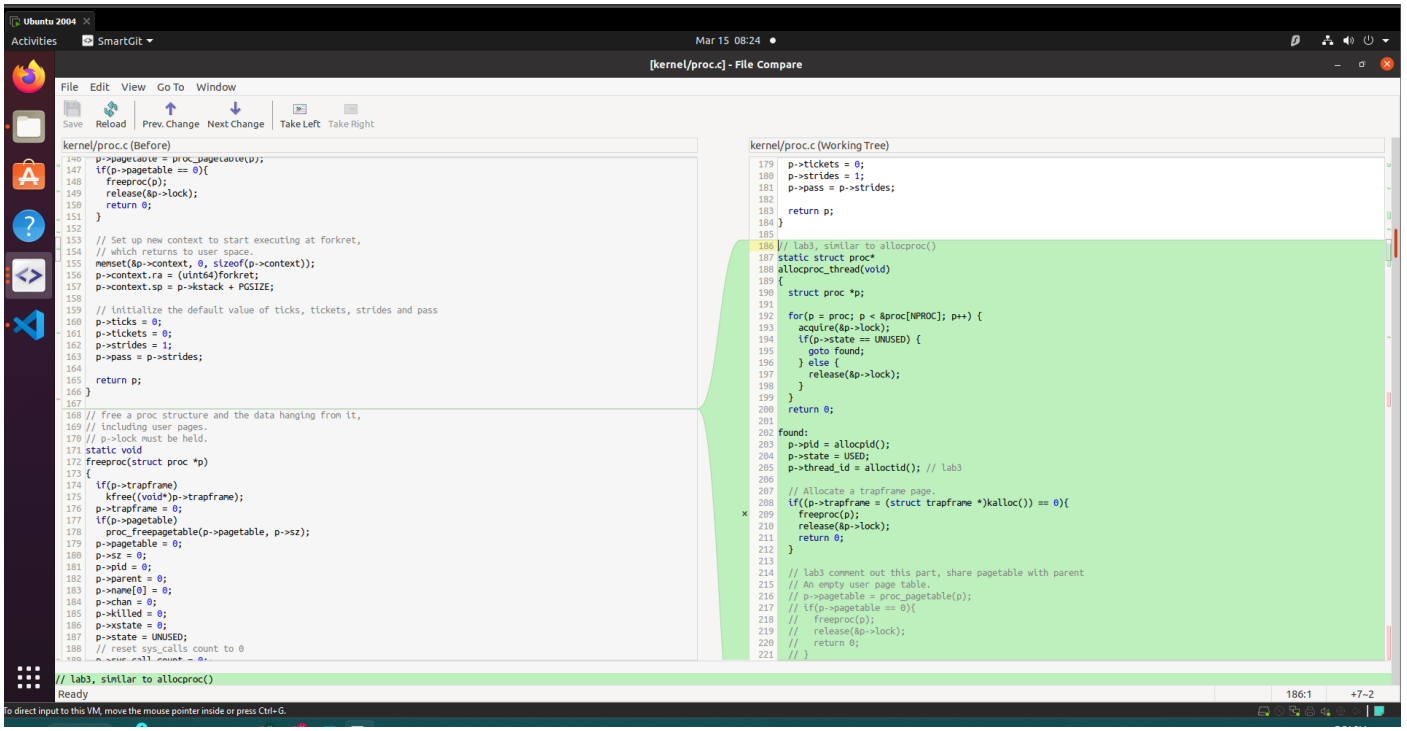

SmartGit
Mar 15 08:22
[Makefile] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
Makefile (Before)
Makefile (Working Tree)
Ready
75:1
+1~3

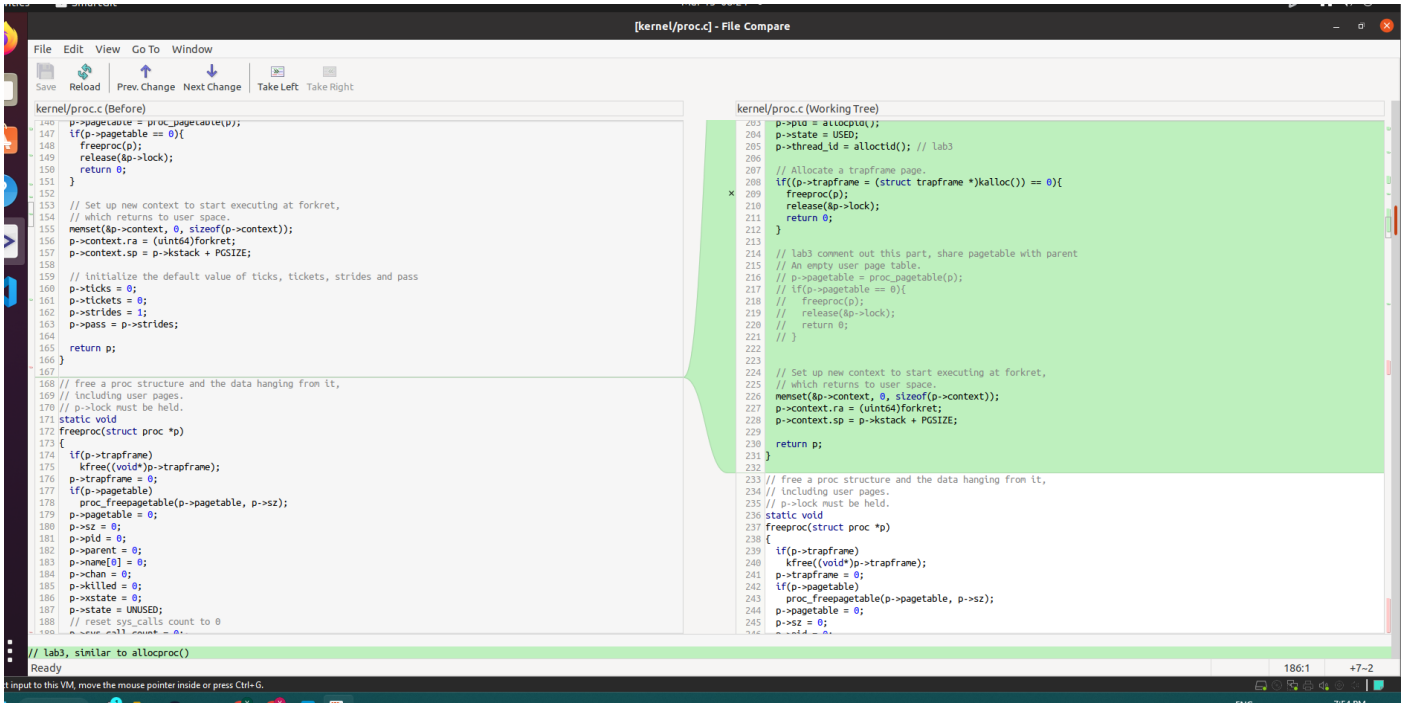

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
#include "types.h"
#include "param.h"
#include "memlayout.h"
#include "riscv.h"
#include "spinlock.h"
#include "proc.h"
#include "defs.h"
struct cpu cpus[NCPU];
struct proc proc[NPROC];
struct proc *initproc;
int nextpid = 1;
struct spinlock pid_lock;
int nexttid = 1; // lab3
struct spinlock tid_lock; // lab3
extern void forkret(void);
static void freeproc(struct proc *p);
extern char trampoline[]; // trampoline.S
// helps ensure that wakeups of wait()ing
// parents are not lost. helps obey the
// memory model when using p->parent.
// must be acquired before any p->lock.
struct spinlock wait_lock;
// psuedo random number generator
// https://stackoverflow.com/a/7603688
unsigned short lfsr = 0xACE1u;
unsigned short bit;
unsigned short rand()
{
bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1;
return lfsr = (lfsr >> 1) | (bit << 15);
}
// Allocate a page for each process's kernel stack.
// Map it high in memory, followed by an invalid
// guard page.
void
proc_mapstacks(pagetable_t kpgtbl)
int nexttid = 1; // lab3
Ready
16:1
+7~2

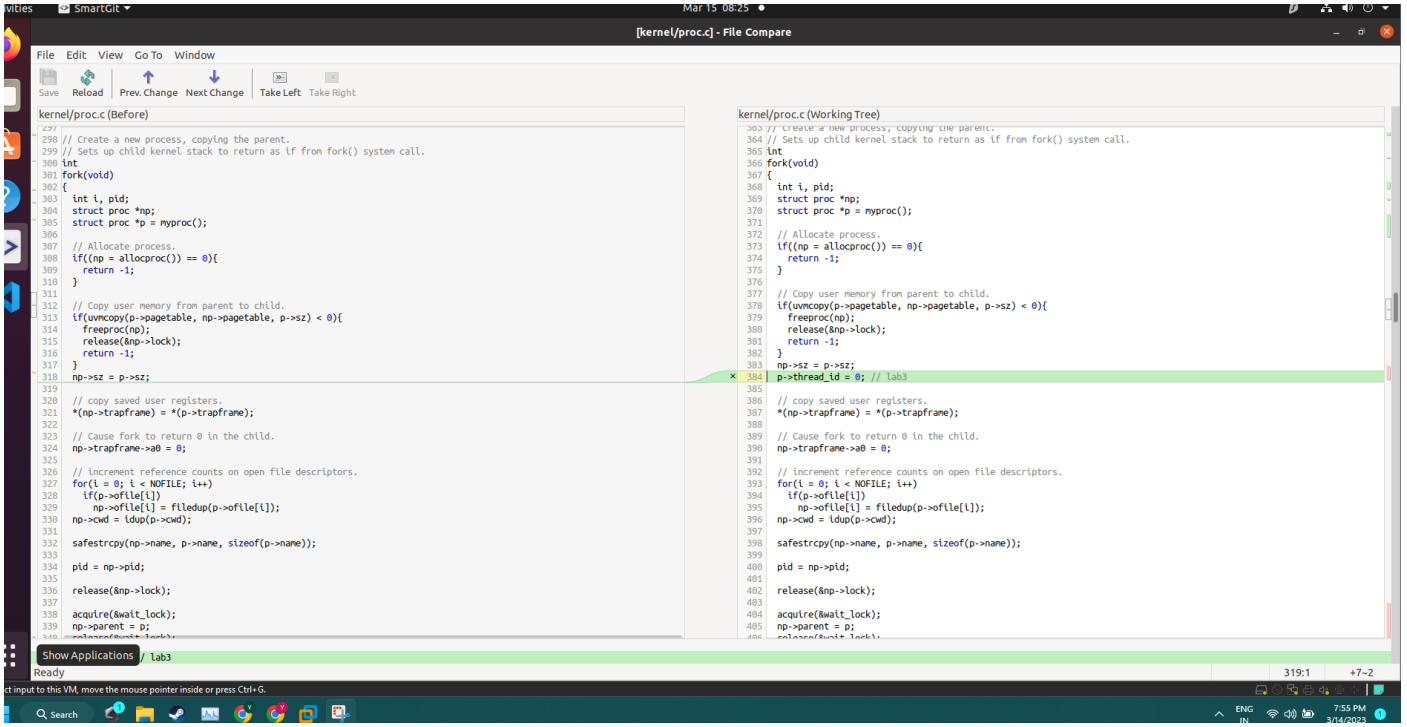

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
proc_mapstacks(pagetable_t kpgtbl)
{
struct proc *p;
for(p = proc; p < &proc[NPROC]; p++) {
char *pa = kalloc();
if(pa == 0)
panic("kalloc");
uint64 va = KSTACK((int) (p - proc));
kvmmap(kpgtbl, va, (uint64)pa, PGSIZE, PTE_R | PTE_W);
}
}
// initialize the proc table.
void
procinit(void)
{
struct proc *p;
initlock(&pid_lock, "nextpid");
initlock(&wait_lock, "wait_lock");
initlock(&tid_lock, "nexttid");
for(p = proc; p < &proc[NPROC]; p++) {
initlock(&p->lock, "proc");
p->state = UNUSED;
p->kstack = KSTACK((int) (p - proc));
}
}
// Must be called with interrupts disabled,
// to prevent race with process being moved
// to a different CPU.
int
cpuid()
{
int id = r_tp();
return id;
}
// Return this CPU's cpu struct.
// Interrupts must be disabled.
struct cpu*
mycpu(void)
initlock(&tid_lock, "nexttid");
Ready
64:1
+7~2

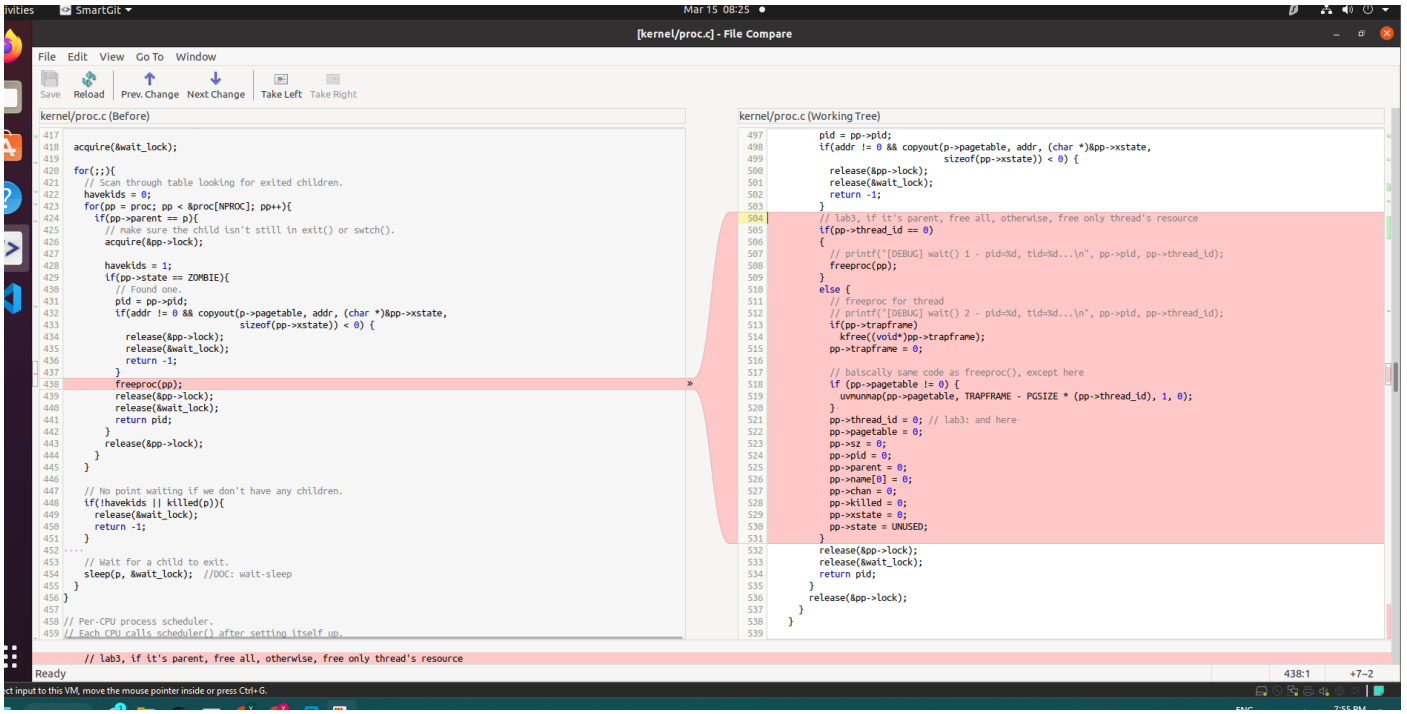

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
// lab3, similar to allocpid()
Ready
115:1
+7−2

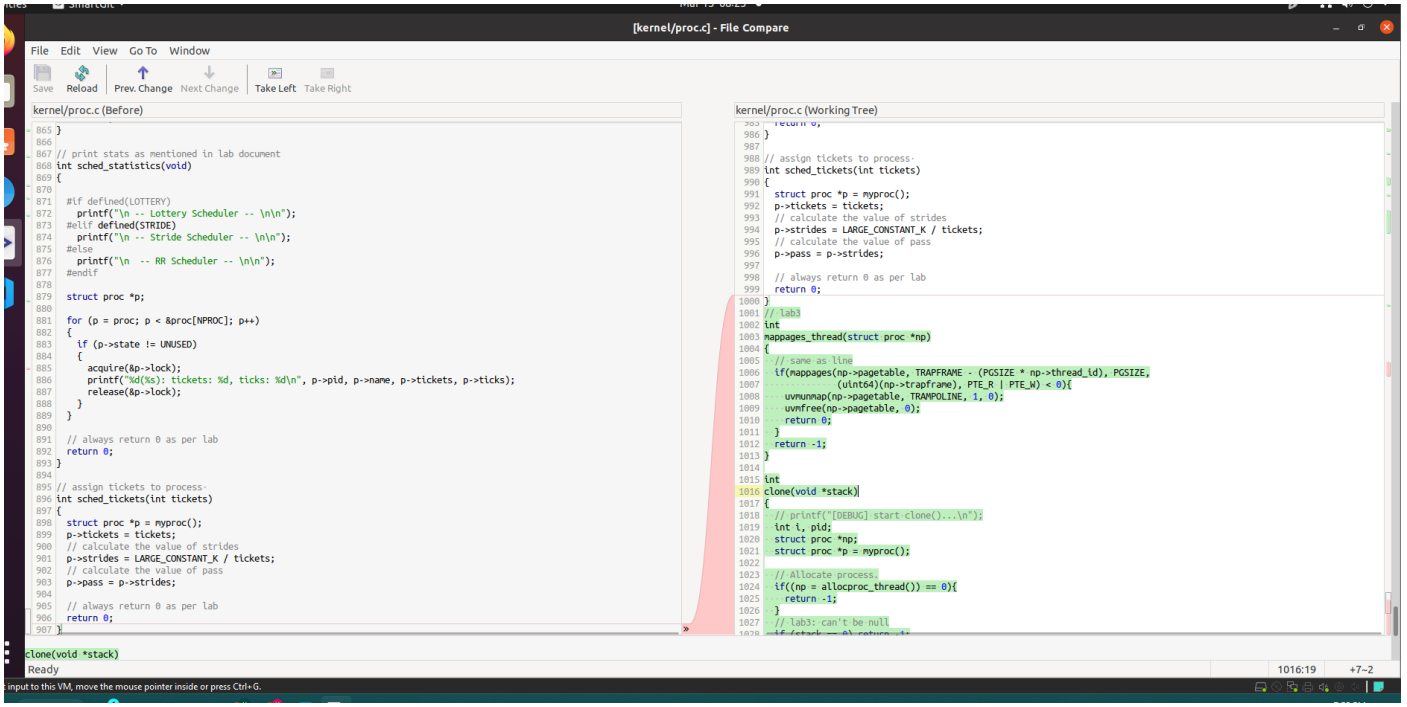

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
p->thread_id = 0; // lab3
Ready
137:1
+7−2

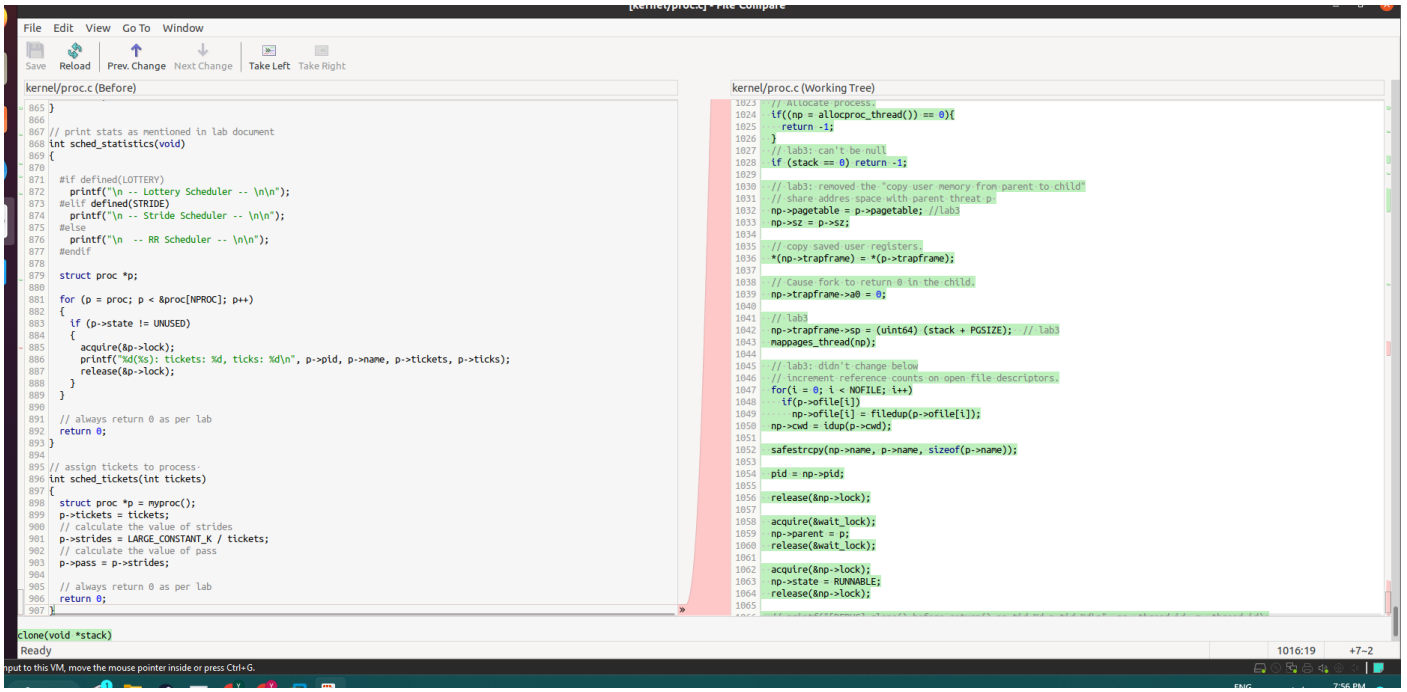

Ubuntu 2004
Activities
SmartGit
Mar 15 08:24
[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
// lab3, similar to allocproc()
Ready
186:1
+7~2
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

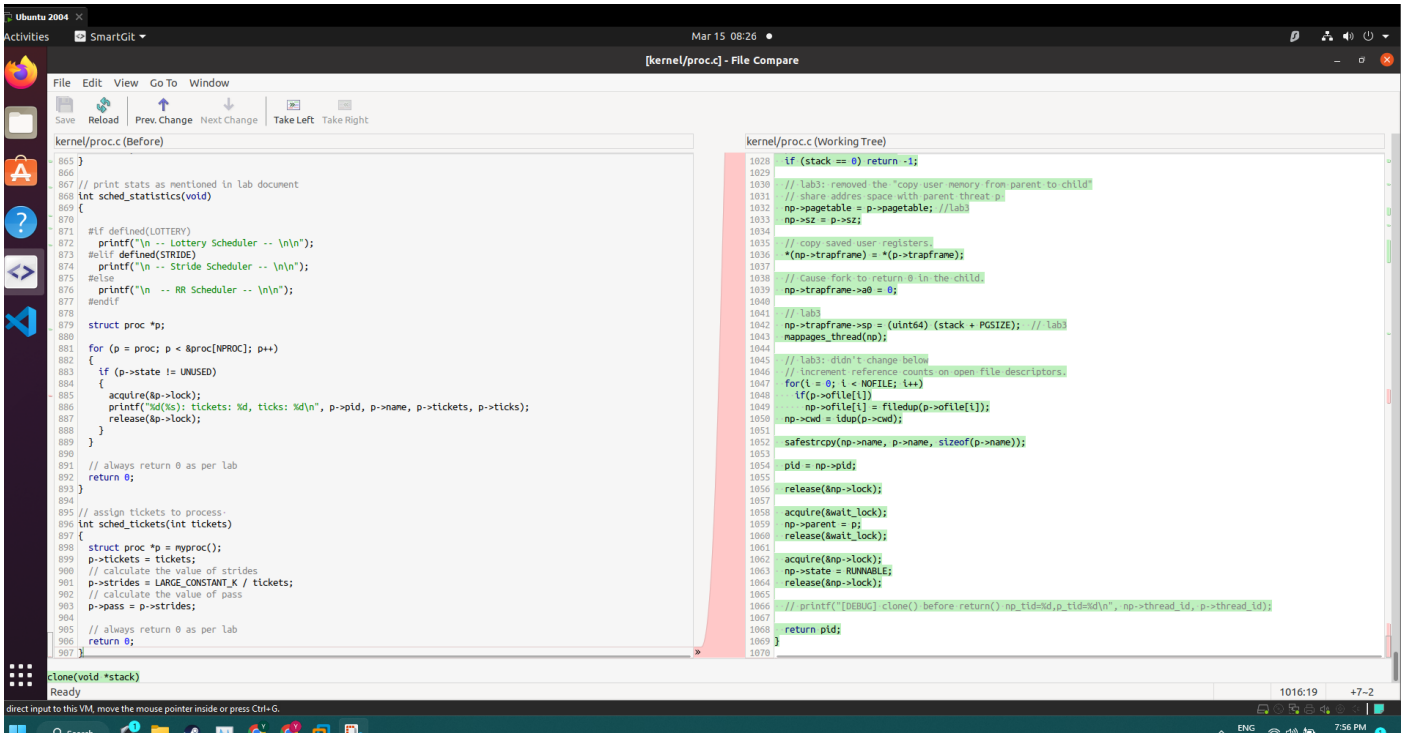

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
// lab3, similar to allocproc()
Ready
186:1
+7~2

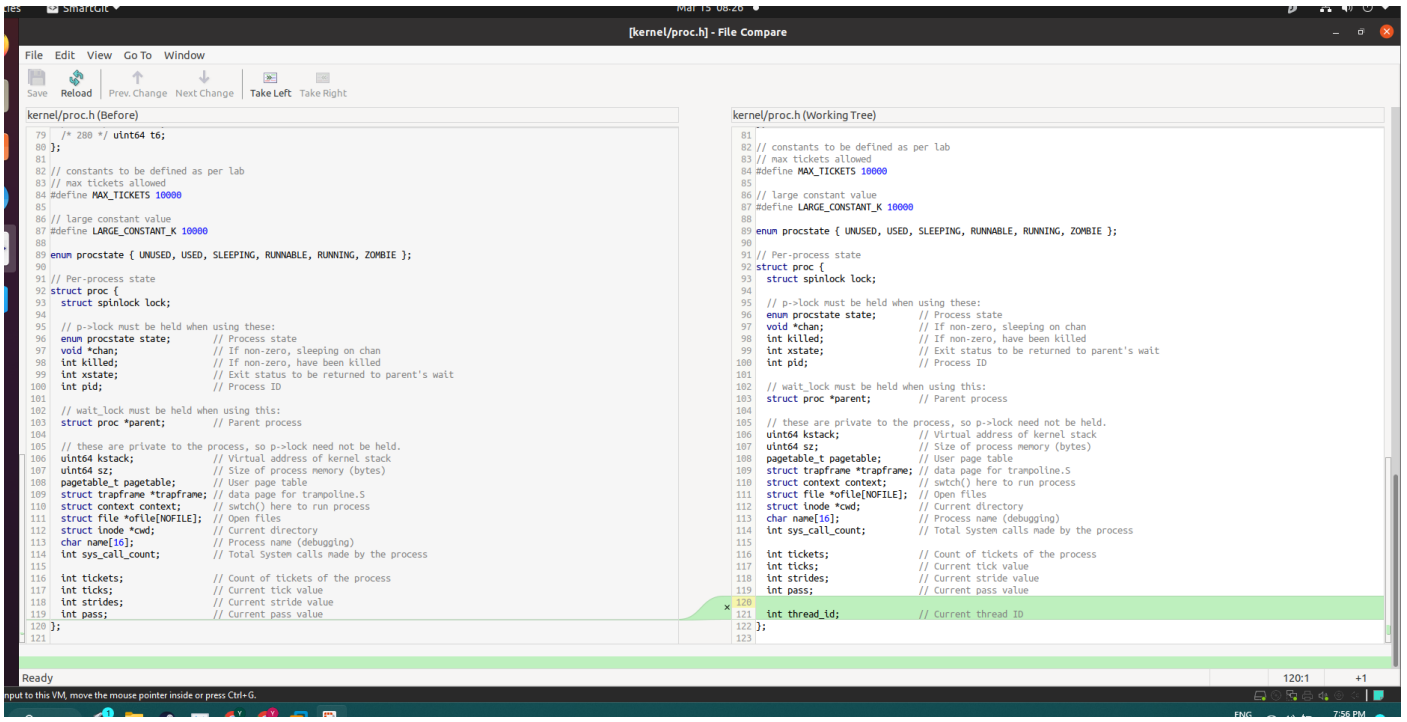

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
// Create a new process, copying the parent.
// Sets up child kernel stack to return as if from fork() system call.
int
fork(void)
{
int i, pid;
struct proc *np;
struct proc *p = myproc();
// Allocate process.
if((np = allocproc()) == 0){
return -1;
}
// Copy user memory from parent to child.
if(uvmcopy(p->pagetable, np->pagetable, p->sz) < 0){
freeproc(np);
release(&np->lock);
return -1;
}
np->sz = p->sz;
np->thread_id = 0; // lab3
// copy saved user registers.
*(np->trapframe) = *(p->trapframe);
// Cause fork to return 0 in the child.
np->trapframe->a0 = 0;
// increment reference counts on open file descriptors.
for(i = 0; i < NOFILE; i++)
if(p->ofile[i])
np->ofile[i] = filedup(p->ofile[i]);
np->cwd = idup(p->cwd);
safestrcpy(np->name, p->name, sizeof(p->name));
pid = np->pid;
release(&np->lock);
acquire(&wait_lock);
np->parent = p;
Show Applications
Ready
319:1
+7−2
Search
7:55 PM
3/14/2023

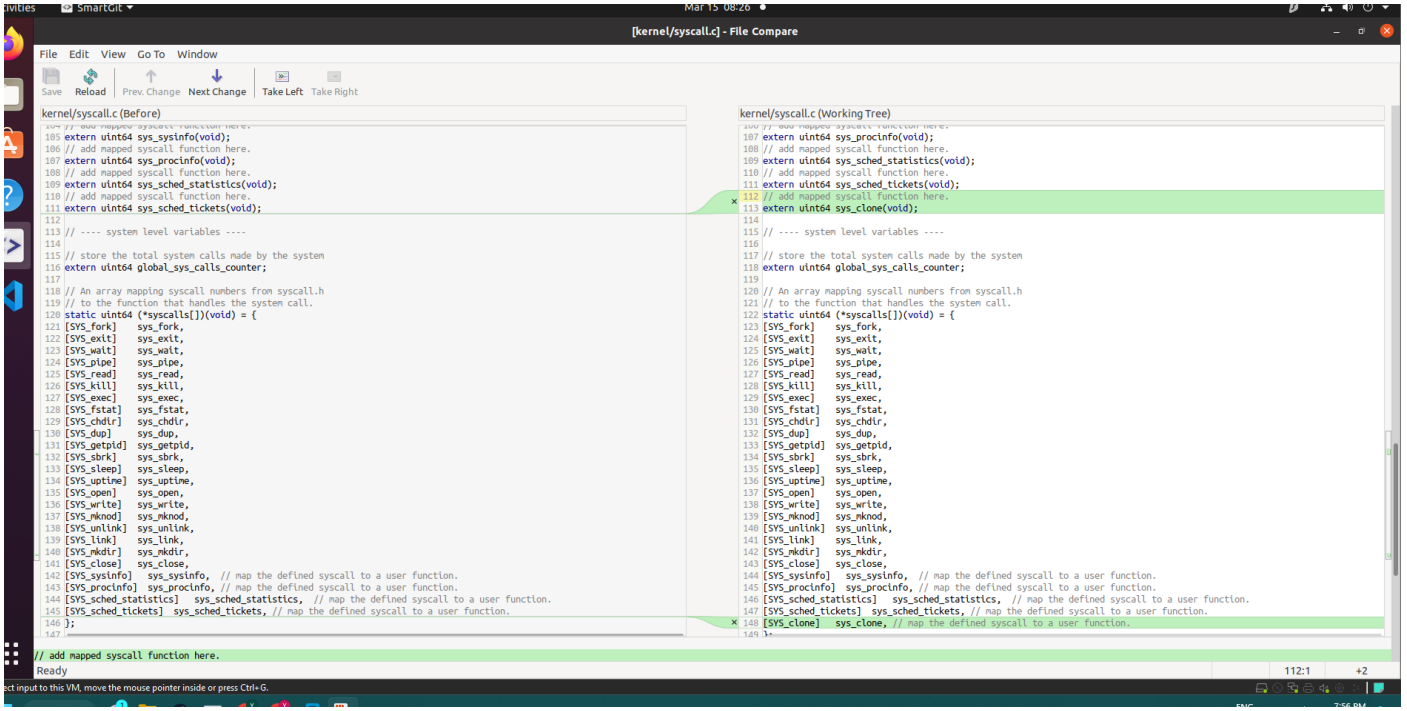

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
acquire(&wait_lock);
for(;;){
// Scan through table looking for exited children.
havekids = 0;
for(pp = proc; pp < &proc[NPROC]; pp++){
if(pp->parent == p){
// make sure the child isn't still in exit() or swtch().
acquire(&pp->lock);
havekids = 1;
if(pp->state == ZOMBIE){
// Found one.
pid = pp->pid;
if(addr != 0 && copyout(p->pagetable, addr, (char *)&pp->xstate,
sizeof(pp->xstate)) < 0) {
release(&pp->lock);
release(&wait_lock);
return -1;
}
freeproc(pp);
release(&pp->lock);
release(&wait_lock);
return pid;
}
release(&pp->lock);
}
}
// No point waiting if we don't have any children.
if(!havekids || killed(p)){
release(&wait_lock);
return -1;
}
// Wait for a child to exit.
sleep(p, &wait_lock); //DOC: wait-sleep
}
}
// Per-CPU process scheduler.
// Each CPU calls scheduler() after setting itself up.
pid = pp->pid;
if(addr != 0 && copyout(p->pagetable, addr, (char *)&pp->xstate,
sizeof(pp->xstate)) < 0) {
release(&pp->lock);
release(&wait_lock);
return -1;
}
// lab3, if it's parent, free all, otherwise, free only thread's resource
if(pp->thread_id == 0)
{
// printf("[DEBUG] wait() 1 - pid=%d, tid=%d...\n", pp->pid, pp->thread_id);
freeproc(pp);
}
else {
// freeproc for thread
// printf("[DEBUG] wait() 2 - pid=%d, tid=%d...\n", pp->pid, pp->thread_id);
if(pp->trapframe)
kfree((void*)pp->trapframe);
pp->trapframe = 0;
// baiscally same code as freeproc(), except here
if (pp->pagetable != 0) {
uvmunmap(pp->pagetable, TRAPFRAME - PGSIZE * (pp->thread_id), 1, 0);
}
pp->thread_id = 0; // lab3: and here
pp->pagetable = 0;
pp->sz = 0;
pp->pid = 0;
pp->parent = 0;
pp->name[0] = 0;
pp->chan = 0;
pp->killed = 0;
pp->xstate = 0;
pp->state = UNUSED;
}
release(&pp->lock);
release(&wait_lock);
return pid;
}
release(&pp->lock);
}
}
// lab3, if it's parent, free all, otherwise, free only thread's resource
Ready
438:1
+7−2

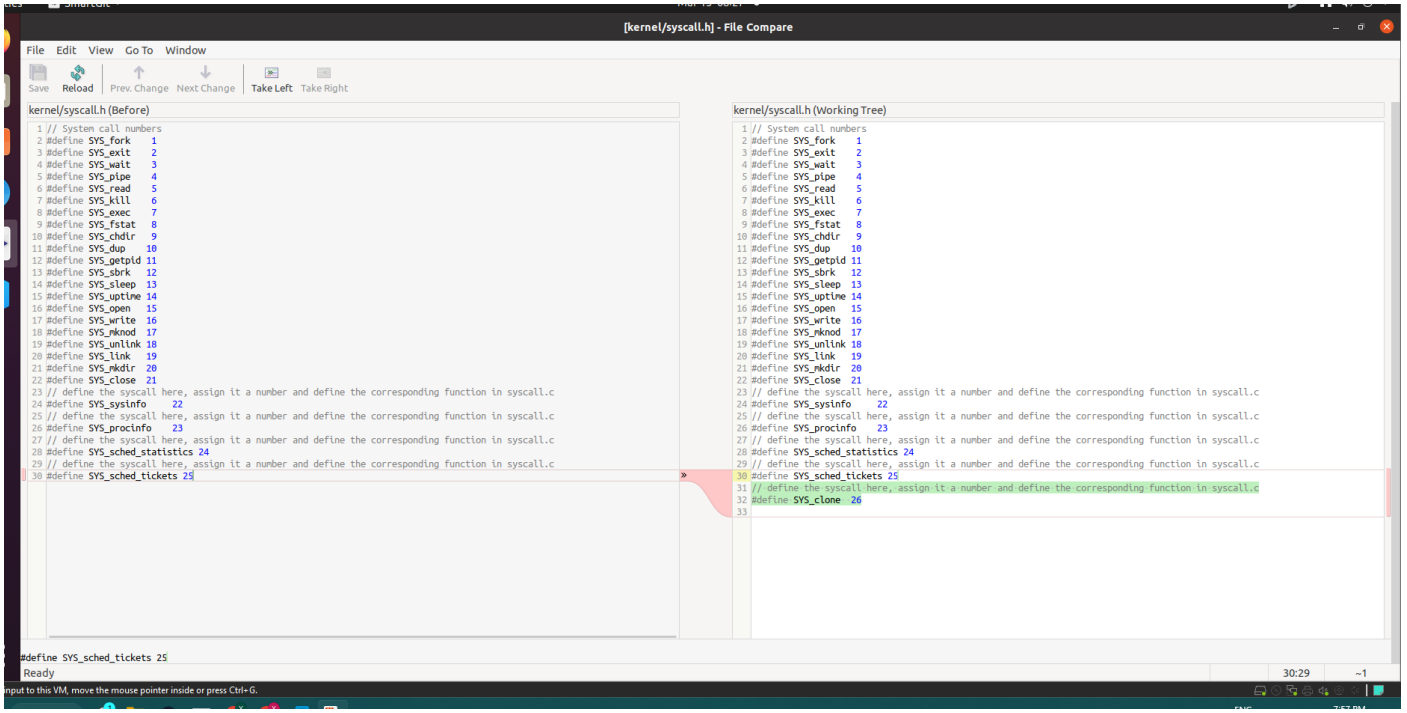

[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
clone(void *stack)
Ready
1016:19
+7−2
input to this VM, move the mouse pointer inside or press Ctrl+G.

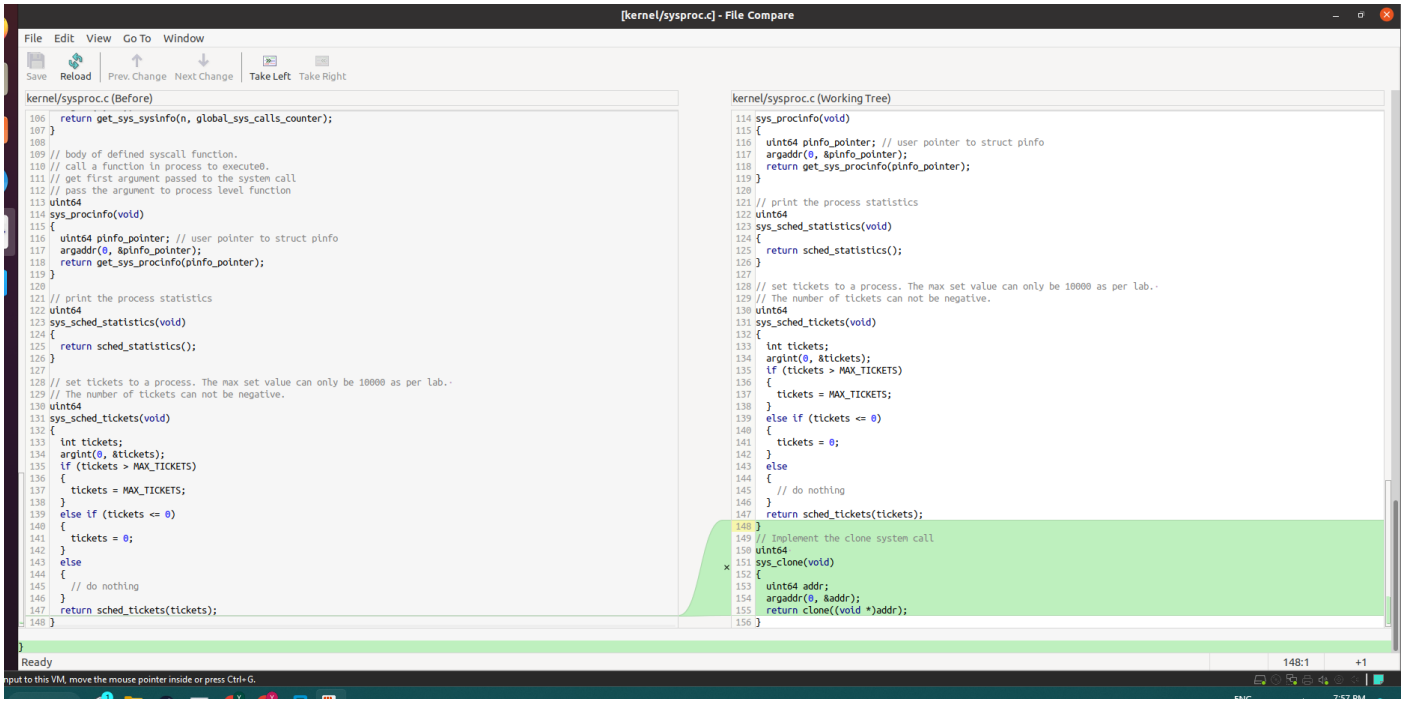

File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
clone(void *stack)
Ready
1016:19
+7−2
input to this VM, move the mouse pointer inside or press Ctrl+G.

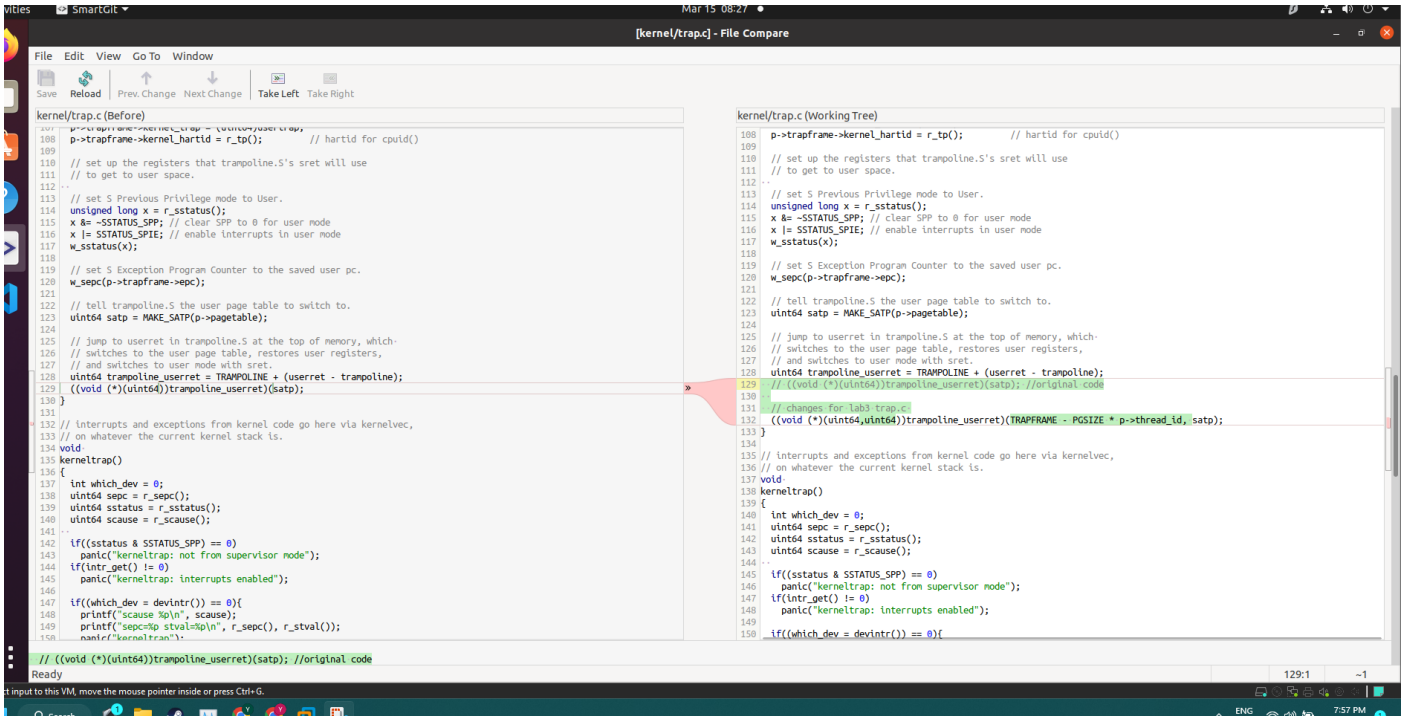

Ubuntu 2004
Activities
SmartGit
Mar 15 08:26
[kernel/proc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.c (Before)
kernel/proc.c (Working Tree)
clone(void *stack)
Ready
1016:19
+7~2
direct input to this VM, move the mouse pointer inside or press Ctrl+G.
ENG
7:56 PM

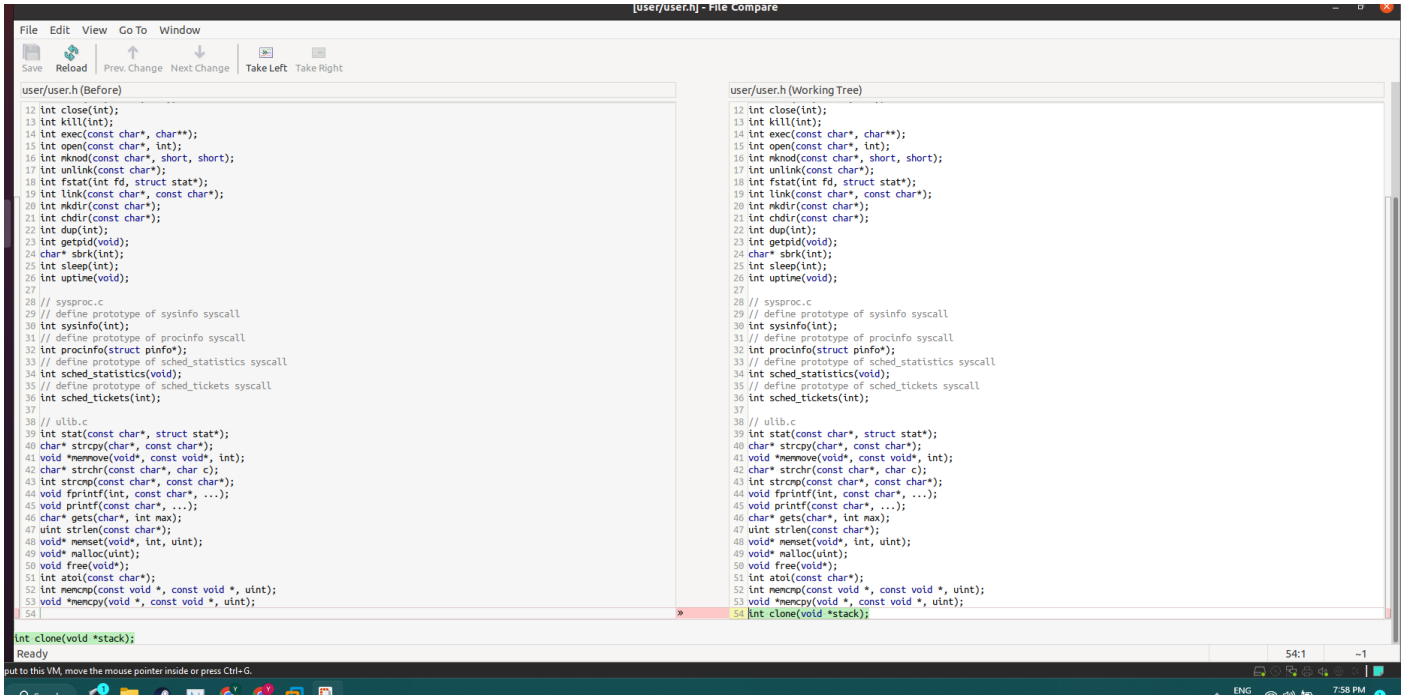

SmartGit
[kernel/proc.h] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/proc.h (Before)
kernel/proc.h (Working Tree)
Ready
120:1
+1
nput to this VM, move the mouse pointer inside or press Ctrl+G.
ENG
7:56 PM

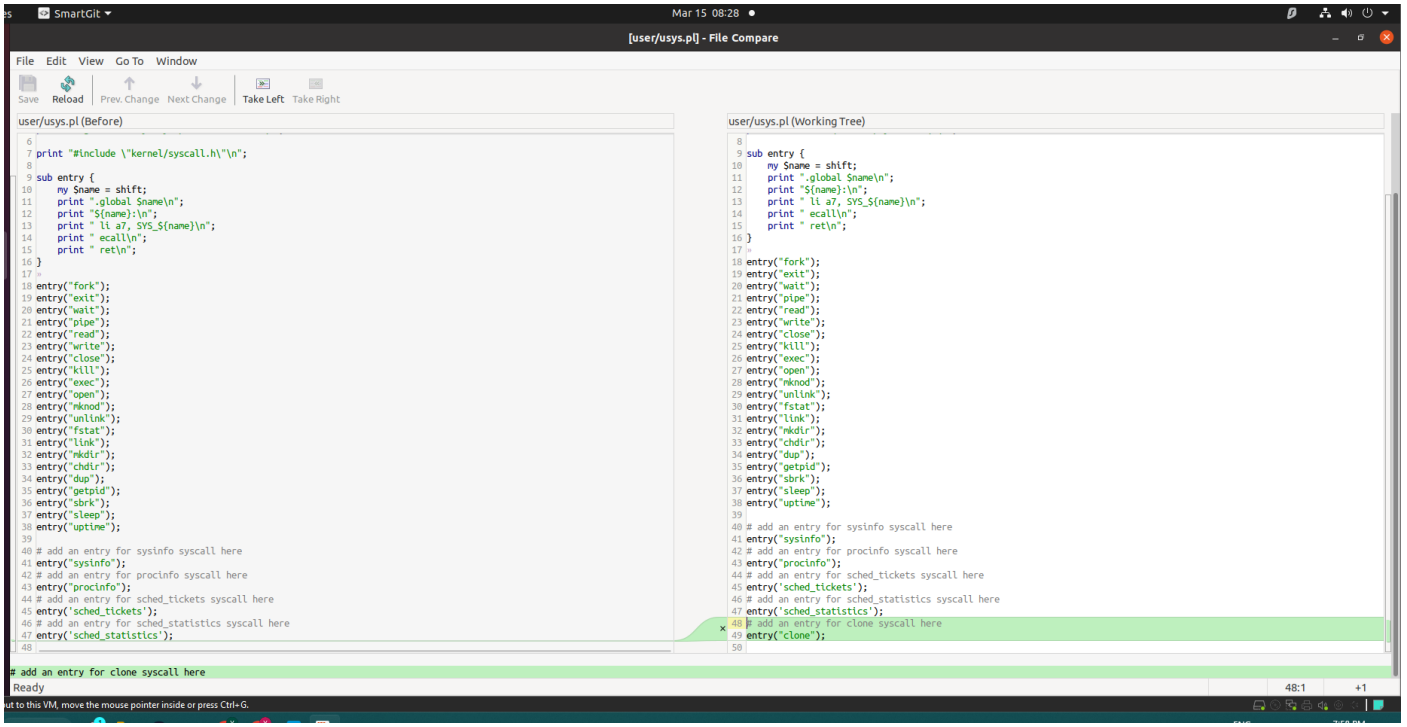

[kernel/syscall.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/syscall.c (Before)
kernel/syscall.c (Working Tree)
// add mapped syscall function here.
Ready
112:1 +2

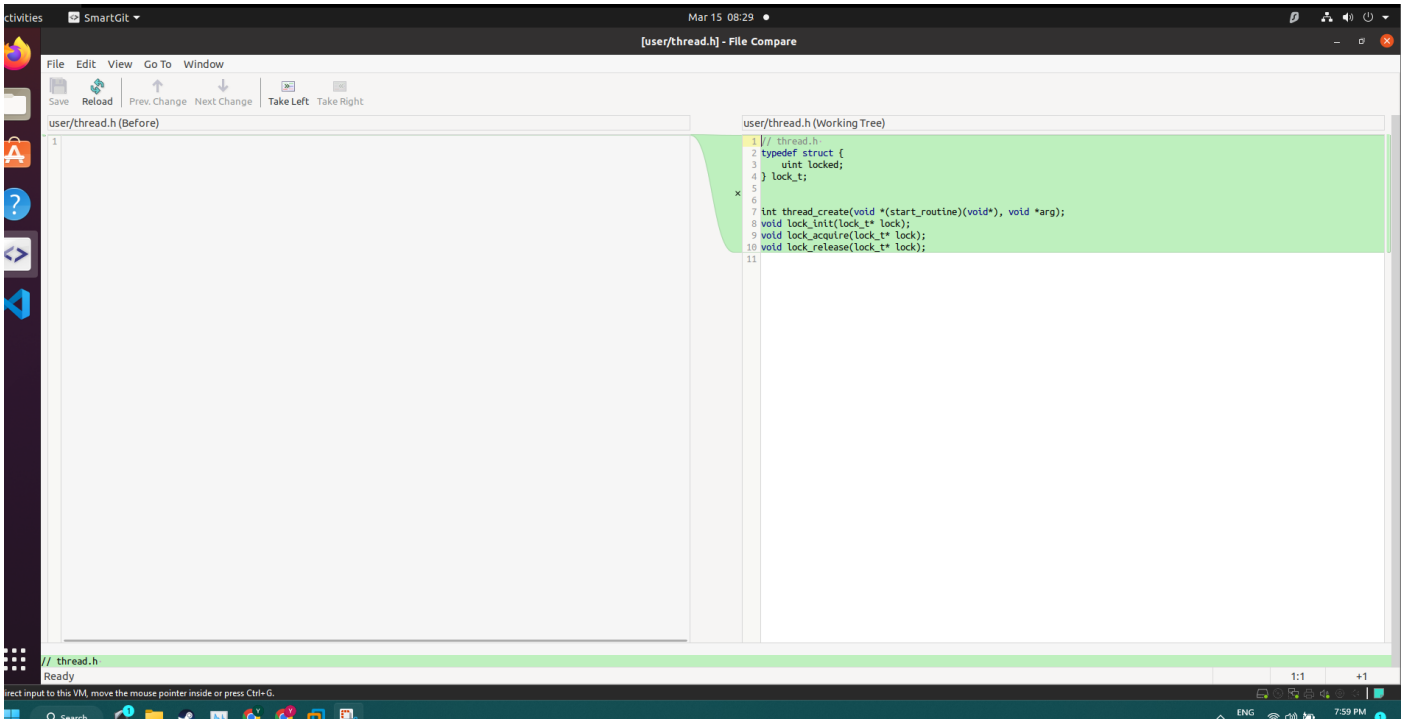

[kernel/syscall.h] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/syscall.h (Before)
kernel/syscall.h (Working Tree)
#define SYS_sched_tickets 25
Ready
30:29 ~1

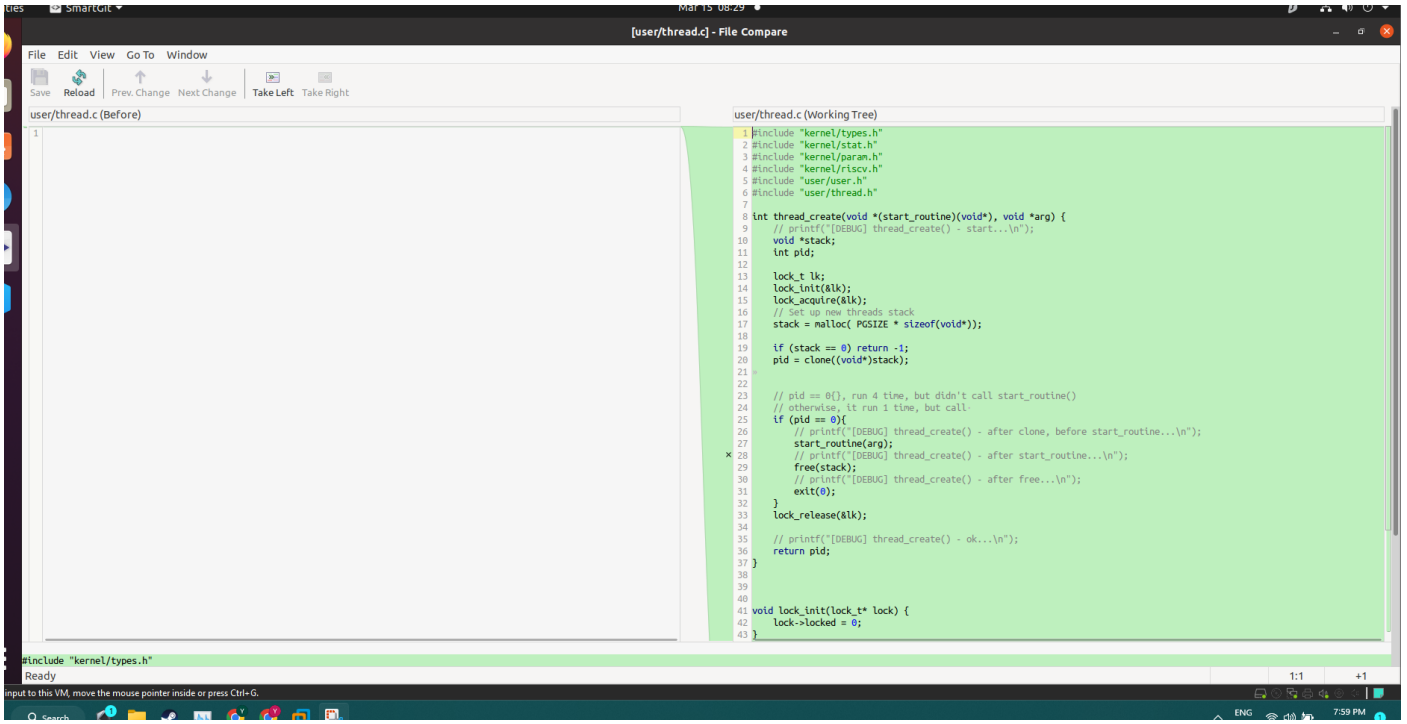

[kernel/sysproc.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/sysproc.c (Before)
kernel/sysproc.c (Working Tree)
Ready
148:1 +1

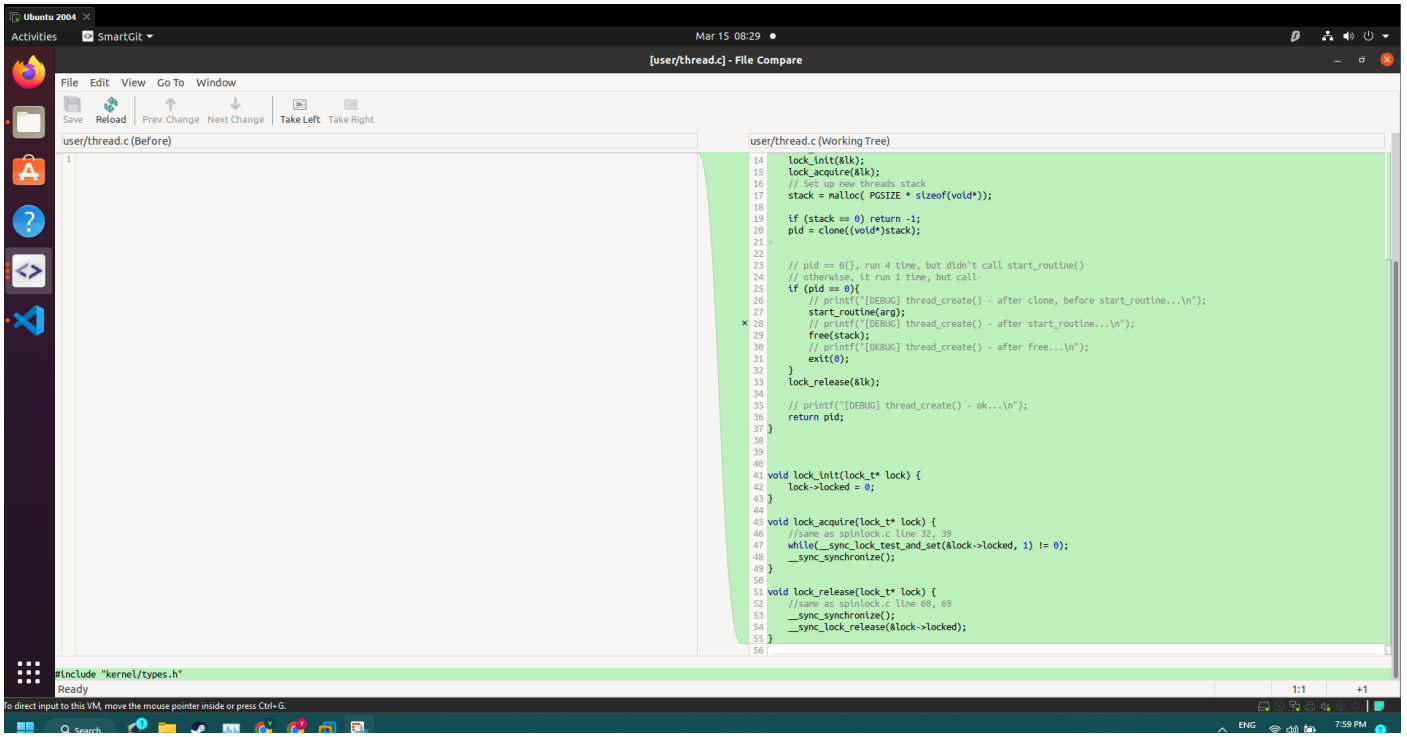

[kernel/trap.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
kernel/trap.c (Before)
kernel/trap.c (Working Tree)
Ready
129:1 -1

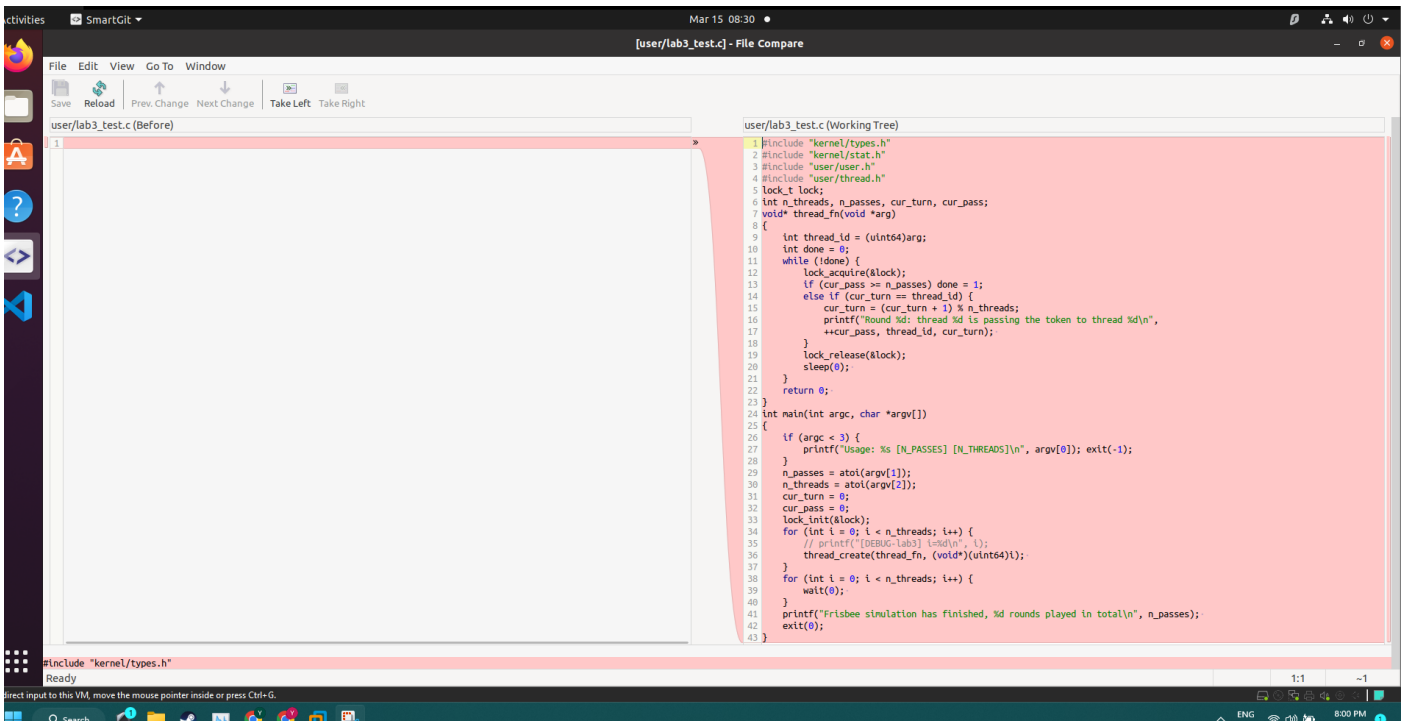

[user/user.h] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/user.h (Before)
user/user.h (Working Tree)
int close(int);
int kill(int);
int exec(const char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
// sysproc.c
// define prototype of sysinfo syscall
int sysinfo(int);
// define prototype of procinfo syscall
int procinfo(struct pinfo*);
// define prototype of sched_statistics syscall
int sched_statistics(void);
// define prototype of sched_tickets syscall
int sched_tickets(int);
// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void fprintf(int, const char*, ...);
void printf(const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
int memcmp(const void *, const void *, uint);
void *memcpy(void *, const void *, uint);
int clone(void *stack);
Ready
54:1
~1


SmartGit
Mar 15 08:28
[user/usys.pl] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/usys.pl (Before)
user/usys.pl (Working Tree)
print "#include \"kernel/syscall.h\"\n";
sub entry {
my $name = shift;
print ".global $name\n";
print "${name}:\n";
print " li a7, SYS_${name}\n";
print " ecall\n";
print " ret\n";
}
entry("fork");
entry("exit");
entry("wait");
entry("pipe");
entry("read");
entry("write");
entry("close");
entry("kill");
entry("exec");
entry("open");
entry("mknod");
entry("unlink");
entry("fstat");
entry("link");
entry("mkdir");
entry("chdir");
entry("dup");
entry("getpid");
entry("sbrk");
entry("sleep");
entry("uptime");
# add an entry for sysinfo syscall here
entry("sysinfo");
# add an entry for procinfo syscall here
entry("procinfo");
# add an entry for sched_tickets syscall here
entry('sched_tickets');
# add an entry for sched_statistics syscall here
entry('sched_statistics');
# add an entry for clone syscall here
entry("clone");
Ready
48:1
+1


[user/thread.h] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/thread.h (Before)
user/thread.h (Working Tree)
// thread.h
typedef struct {
uint locked;
} lock_t;
int thread_create(void *(start_routine)(void*), void *arg);
void lock_init(lock_t* lock);
void lock_acquire(lock_t* lock);
void lock_release(lock_t* lock);
// thread.h
Ready
1:1
+1


[user/thread.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/thread.c (Before)
user/thread.c (Working Tree)
#include "kernel/types.h"
#include "kernel/stat.h"
#include "kernel/param.h"
#include "kernel/riscv.h"
#include "user/user.h"
#include "user/thread.h"
int thread_create(void *(start_routine)(void*), void *arg) {
// printf("[DEBUG] thread_create() - start...\n");
void *stack;
int pid;
lock_t lk;
lock_init(&lk);
lock_acquire(&lk);
// Set up new threads stack
stack = malloc(PGSIZE * sizeof(void*));
if (stack == 0) return -1;
pid = clone((void*)stack);
// pid == 0{}, run 4 time, but didn't call start_routine()
// otherwise, it run 1 time, but call
if (pid == 0){
// printf("[DEBUG] thread_create() - after clone, before start_routine...\n");
start_routine(arg);
// printf("[DEBUG] thread_create() - after start_routine...\n");
free(stack);
// printf("[DEBUG] thread_create() - after free...\n");
exit(0);
}
lock_release(&lk);
// printf("[DEBUG] thread_create() - ok...\n");
return pid;
}
void lock_init(lock_t* lock) {
lock->locked = 0;
}
#include "kernel/types.h"
Ready
1:1
+1


Ubuntu 2004
Activities
SmartGit
Mar 15 08:29
[user/thread.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/thread.c (Before)
user/thread.c (Working Tree)
lock_init(&lk);
lock_acquire(&lk);
// Set up new threads stack
stack = malloc(PGSIZE * sizeof(void*));
if (stack == 0) return -1;
pid = clone((void*)stack);
// pid == 0{}, run 4 time, but didn't call start_routine()
// otherwise, it run 1 time, but call
if (pid == 0){
// printf("[DEBUG] thread_create() - after clone, before start_routine...\n");
start_routine(arg);
// printf("[DEBUG] thread_create() - after start_routine...\n");
free(stack);
// printf("[DEBUG] thread_create() - after free...\n");
exit(0);
}
lock_release(&lk);
// printf("[DEBUG] thread_create() - ok...\n");
return pid;
}
void lock_init(lock_t* lock) {
lock->locked = 0;
}
void lock_acquire(lock_t* lock) {
//same as spinlock.c line 32, 39
while(__sync_lock_test_and_set(&lock->locked, 1) != 0);
__sync_synchronize();
}
void lock_release(lock_t* lock) {
//same as spinlock.c line 60, 69
__sync_synchronize();
__sync_lock_release(&lock->locked);
}
#include "kernel/types.h"
Ready
1:1
+1
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
ENG
7:59 PM


Activities
SmartGit
Mar 15 08:30
[user/lab3_test.c] - File Compare
File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right
user/lab3_test.c (Before)
user/lab3_test.c (Working Tree)
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"
#include "user/thread.h"
lock_t lock;
int n_threads, n_passes, cur_turn, cur_pass;
void* thread_fn(void *arg)
{
int thread_id = (uint64)arg;
int done = 0;
while (!done) {
lock_acquire(&lock);
if (cur_pass >= n_passes) done = 1;
else if (cur_turn == thread_id) {
cur_turn = (cur_turn + 1) % n_threads;
printf("Round %d: thread %d is passing the token to thread %d\n",
++cur_pass, thread_id, cur_turn);
}
lock_release(&lock);
sleep(0);
}
return 0;
}
int main(int argc, char *argv[])
{
if (argc < 3) {
printf("Usage: %s [N_PASSES] [N_THREADS]\n", argv[0]); exit(-1);
}
n_passes = atoi(argv[1]);
n_threads = atoi(argv[2]);
cur_turn = 0;
cur_pass = 0;
lock_init(&lock);
for (int i = 0; i < n_threads; i++) {
// printf("[DEBUG-lab3] i=%d\n", i);
thread_create(thread_fn, (void*)(uint64)i);
}
for (int i = 0; i < n_threads; i++) {
wait(0);
}
printf("Frisbee simulation has finished, %d rounds played in total\n", n_passes);
exit(0);
}
#include "kernel/types.h"
Ready
1:1
-1
direct input to this VM, move the mouse pointer inside or press Ctrl+G.
ENG
8:00 PM