به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس آزمایشگاه سیستم عامل

پروژه پنجم

نام و نام خانوادگی	دانیال سعیدی(810198571) سروش صادقیان(810898048) محمد قره حسنلو(810198461)
تاریخ ارسال گزارش	
ریپو گیتهاب	
آخرین Commit ID	

فهرست گزارش سوالات

سوال 1

در لینوکس، هسته از نواحی virtual memory پیگیری memory mapping های پردازه اسفاده میکند؛ مثلا یک process یک VMA برای کدش، یک VMA برای هر نوع دیتا، یک VMA برای هر memory mapping در صورت وجود داشتن، داشته باشد. هر VMA شامل تعدادی page هست که که هر کدام یک entry به page table دارد. اما در xv6، از آدرس های مجازی 32 بیتی استفاده میکند که فضای آدرسی مجازی 4 گیگابایتی را ایجاد میکند. همچنین xv6 از ساختار جدول دو سطحی استفاده میکند. xv6 مفهومی از جافظه مجازی ندارد.

سوال 2

چون که از صفحه پردازه ای استفاده میشود که میخواهیم از آن استفاده کنیم و بقیه را که لازمشان نداریم را لود نمیکنیم و اینکه فقط آدرس شروع صفحه را در اینجا نگه داری میکنیم و نیازی به نگه داری همه آذزس های یک صفحه نمیشود و میتوانیم بقیه آدرس ها را با offset دسترسی پیدا کنیم. همچنین در اینجا با mapping کدها به اشتراک گذاشته میشوند که حافظه کمتری را در نتیجه لازم خواهد داشت.

سوال 3

12 بیت برای entry که دو سطح(page directory و page table) داشته و در بیت D باهم متفاوت اند که در page directory یعنی تغییرات در زمان نوشتن صفحه در دیسک اعمال میشود و در page table مقدار آن تاثیری ندارد، 20 بیت برای پوینتر به سطح بعدی هم وجود دارد.

سوال 4

kalloc فضایی از حافظه فیزیکی را تخصیص میدهد

سوال 5

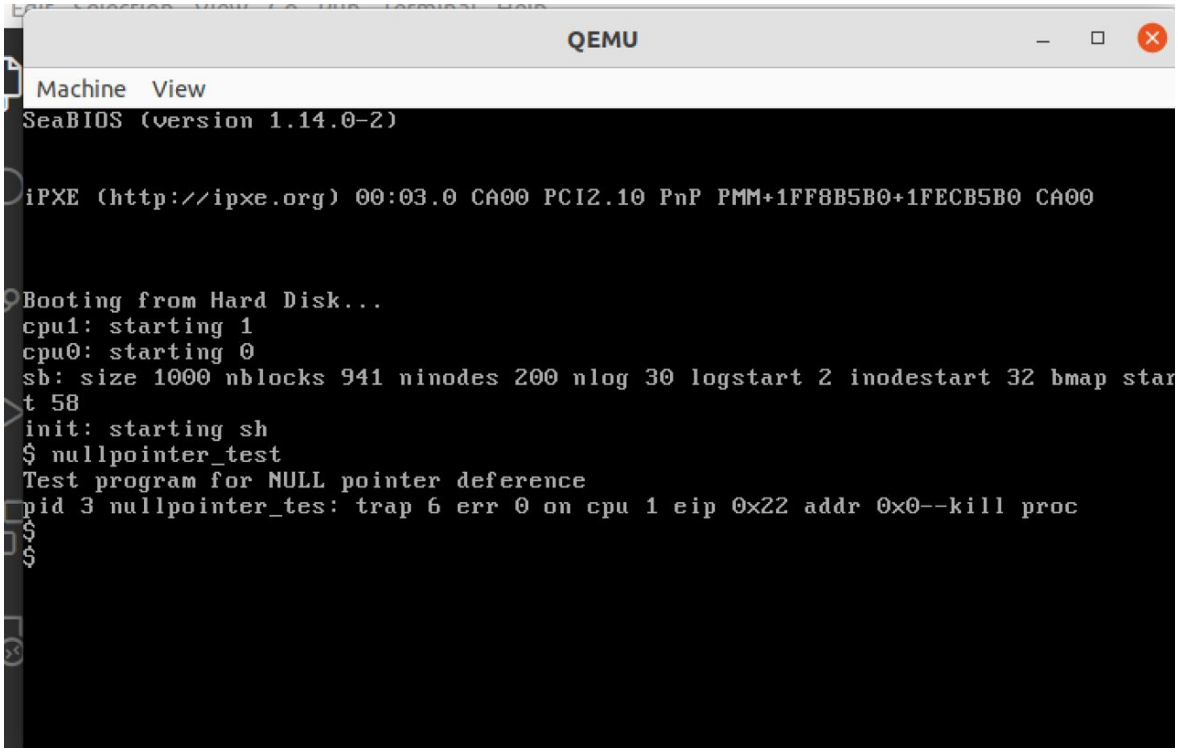
mappages آذرس یک خانه از حفظه فیزیکی و مجازی همراه با سایز را گرفته و صفحه موجود در را در آدرس حافظه مجازی بارگذاری میکند و این کار را با توجه به سایزی که دادیم

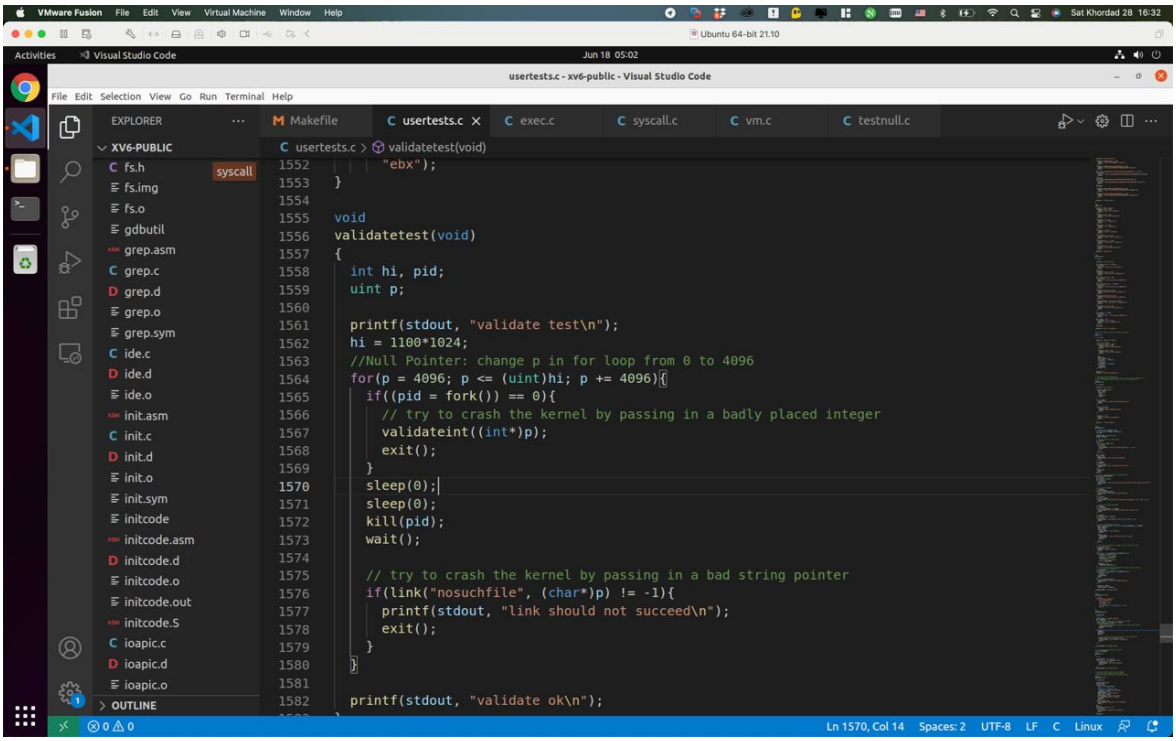
انجام میدهد. دلیل آن به این صورت است که صفحه مربوط به پردازه در حال اجرا را به درستی نشان دهد.

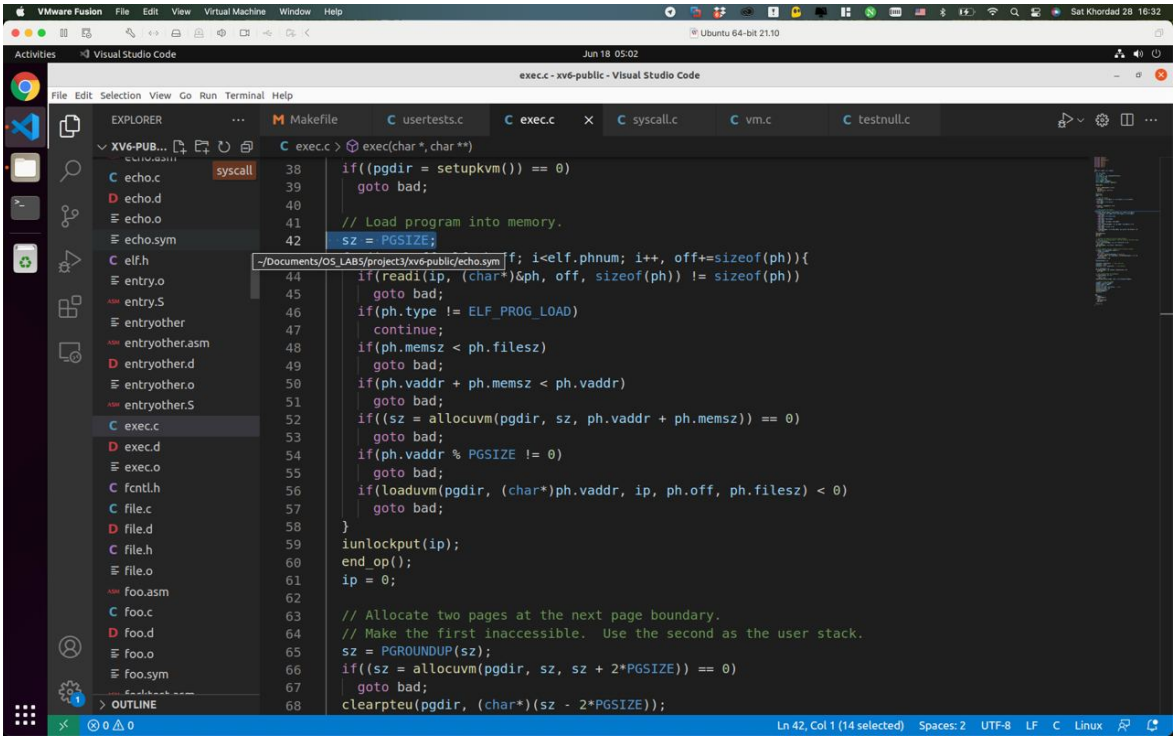
سوال 7

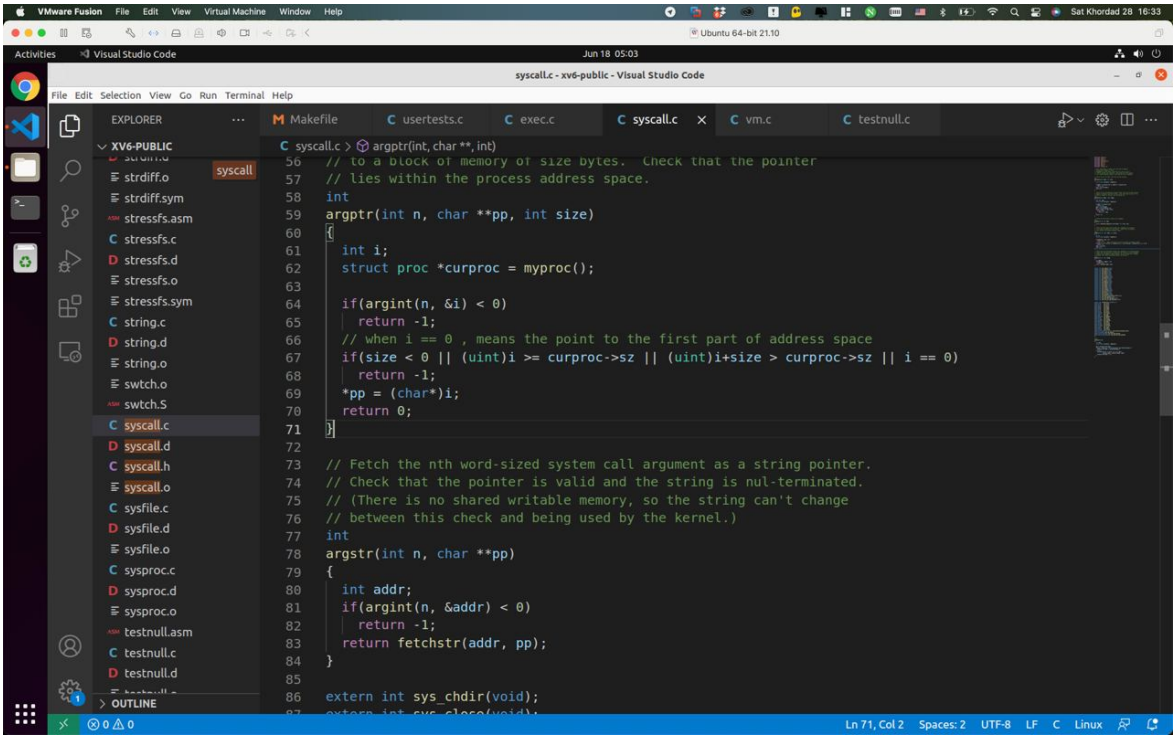
در walkpgdir() آدرس یک خانه حافظه مجازی سطح page directory را میگیرد و آدرس page table که در همان خانه حافظه مجازی است را برمیگرداند.

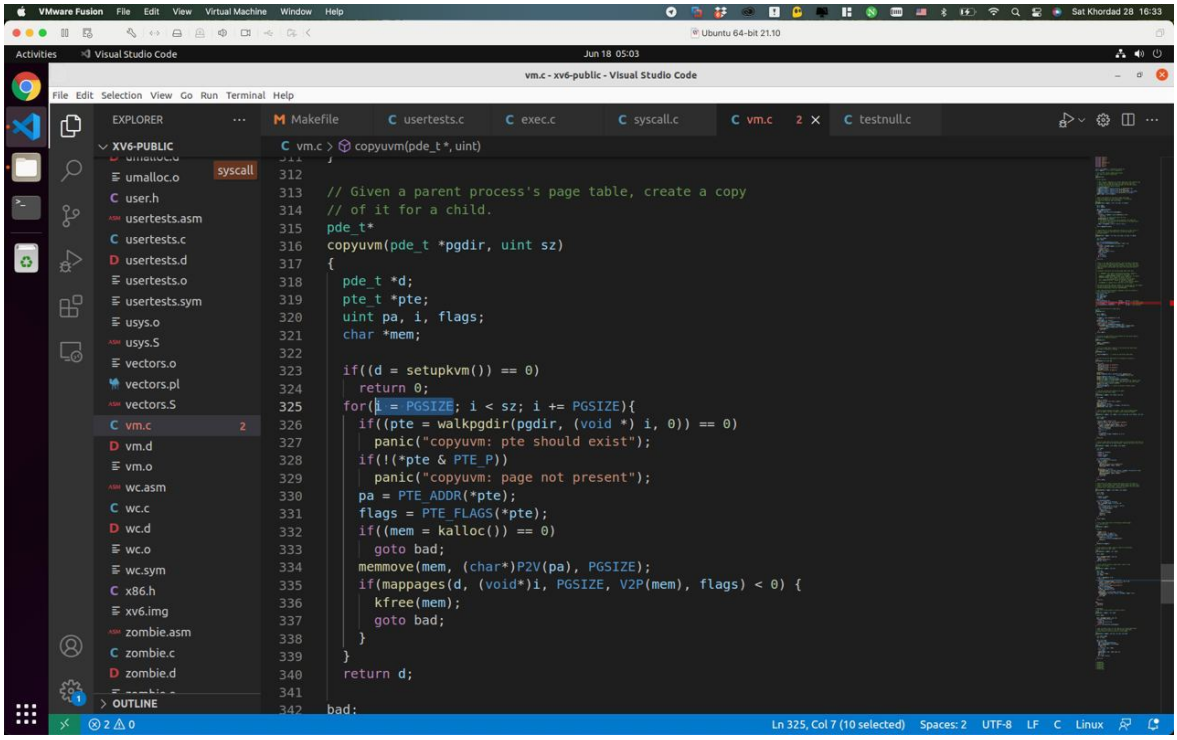
Read Only

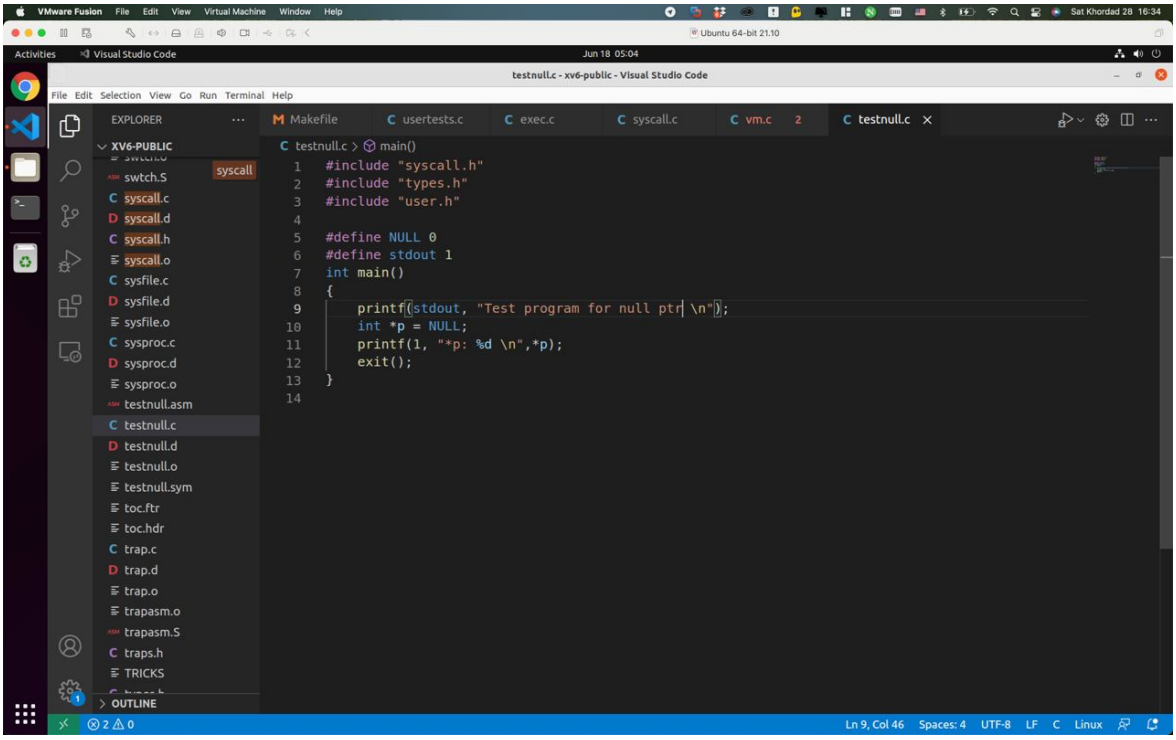


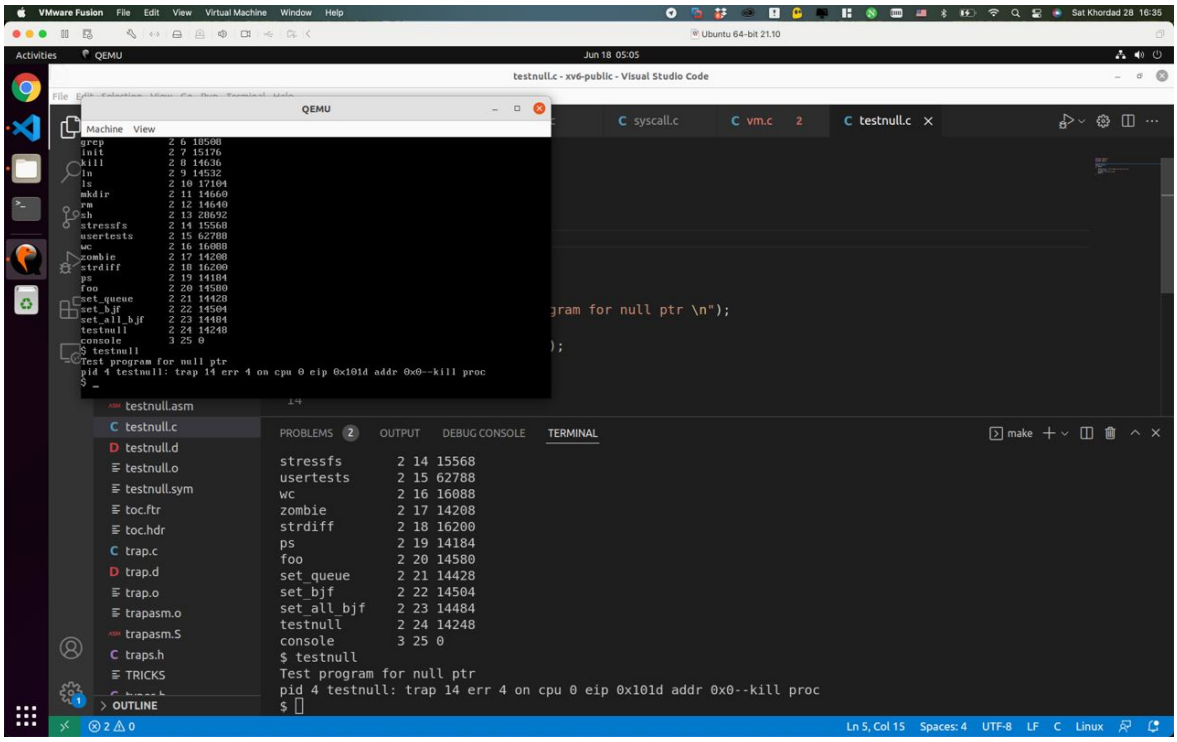

usertests.c - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
Makefile
usertests.c
exec.c
syscall.c
vm.c
testnull.c
usertests.c > validatetest(void)
"ebx");
}
void
validatetest(void)
{
int hi, pid;
uint p;
printf(stdout, "validate test\n");
hi = 1100*1024;
//Null Pointer: change p in for loop from 0 to 4096
for(p = 4096; p <= (uint)hi; p += 4096){
if((pid = fork()) == 0){
// try to crash the kernel by passing in a badly placed integer
validateint((int*)p);
exit();
}
sleep(0);
sleep(0);
kill(pid);
wait();
// try to crash the kernel by passing in a bad string pointer
if(link("nosuchfile", (char*)p) != -1){
printf(stdout, "link should not succeed\n");
exit();
}
}
printf(stdout, "validate ok\n");
OUTLINE
Ln 1570, Col 14
Spaces: 2
UTF-8
LF
C
Linux

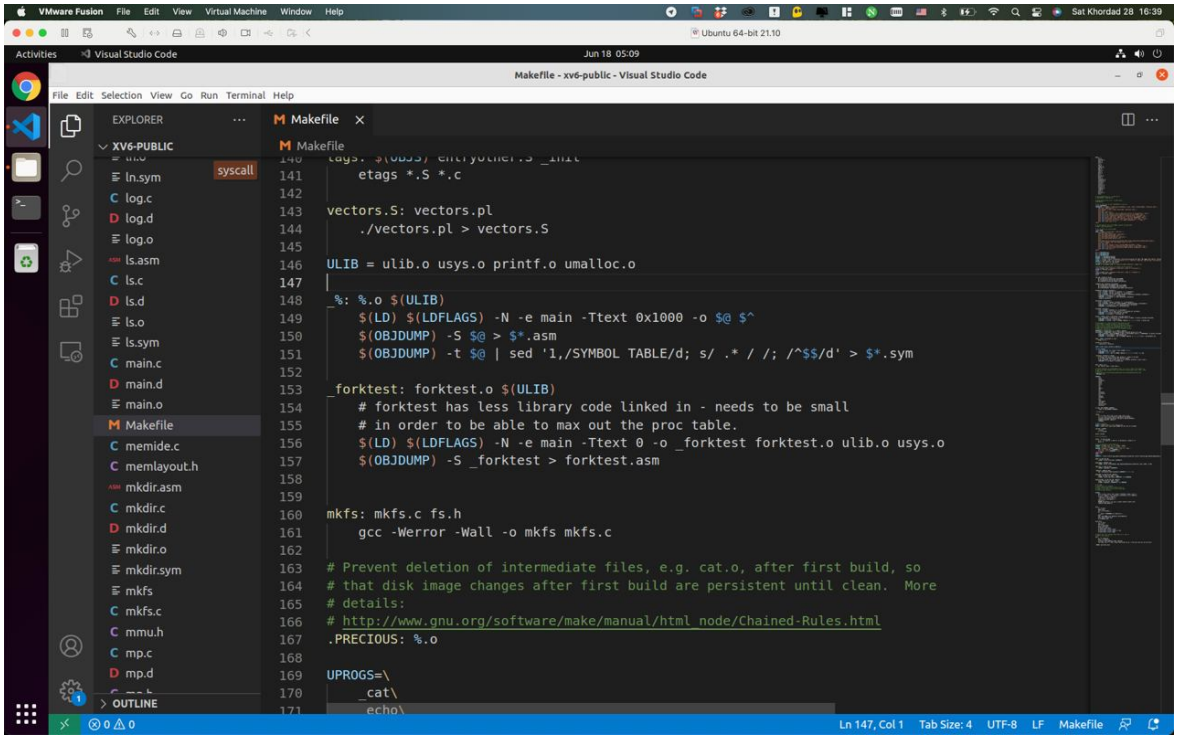

exec.c - xv6-public - Visual Studio Code
EXPLORER
Makefile
usertests.c
exec.c
syscall.c
vm.c
testnull.c
exec.c > exec(char *, char **)
~/Documents/OS_LAB5/project3/xv6-public/echo.sym
if((pgdir = setupkvm()) == 0)
goto bad;
// Load program into memory.
sz = PGSIZE;
f; i<elf.phnum; i++, off+=sizeof(ph)){
if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
goto bad;
if(ph.type != ELF_PROG_LOAD)
continue;
if(ph.memsz < ph.filesz)
goto bad;
if(ph.vaddr + ph.memsz < ph.vaddr)
goto bad;
if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
goto bad;
if(ph.vaddr % PGSIZE != 0)
goto bad;
if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)
goto bad;
}
iunlockput(ip);
end_op();
ip = 0;
// Allocate two pages at the next page boundary.
// Make the first inaccessible. Use the second as the user stack.
sz = PGROUNDUP(sz);
if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
goto bad;
clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
OUTLINE
Ln 42, Col 1 (14 selected)
Spaces: 2
UTF-8
LF
C
Linux


syscall.c - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
Makefile
usertests.c
exec.c
syscall.c
vm.c
testnull.c
syscall.c > argptr(int, char **, int)
// to a block of memory of size bytes. Check that the pointer
// lies within the process address space.
int
argptr(int n, char **pp, int size)
{
int i;
struct proc *curproc = myproc();
if(argint(n, &i) < 0)
return -1;
// when i == 0 , means the point to the first part of address space
if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc->sz || i == 0)
return -1;
pp = (char)i;
return 0;
}
// Fetch the nth word-sized system call argument as a string pointer.
// Check that the pointer is valid and the string is nul-terminated.
// (There is no shared writable memory, so the string can't change
// between this check and being used by the kernel.)
int
argstr(int n, char **pp)
{
int addr;
if(argint(n, &addr) < 0)
return -1;
return fetchstr(addr, pp);
}
extern int sys_chdir(void);
OUTLINE
Ln 71, Col 2
Spaces: 2
UTF-8
LF
C
Linux

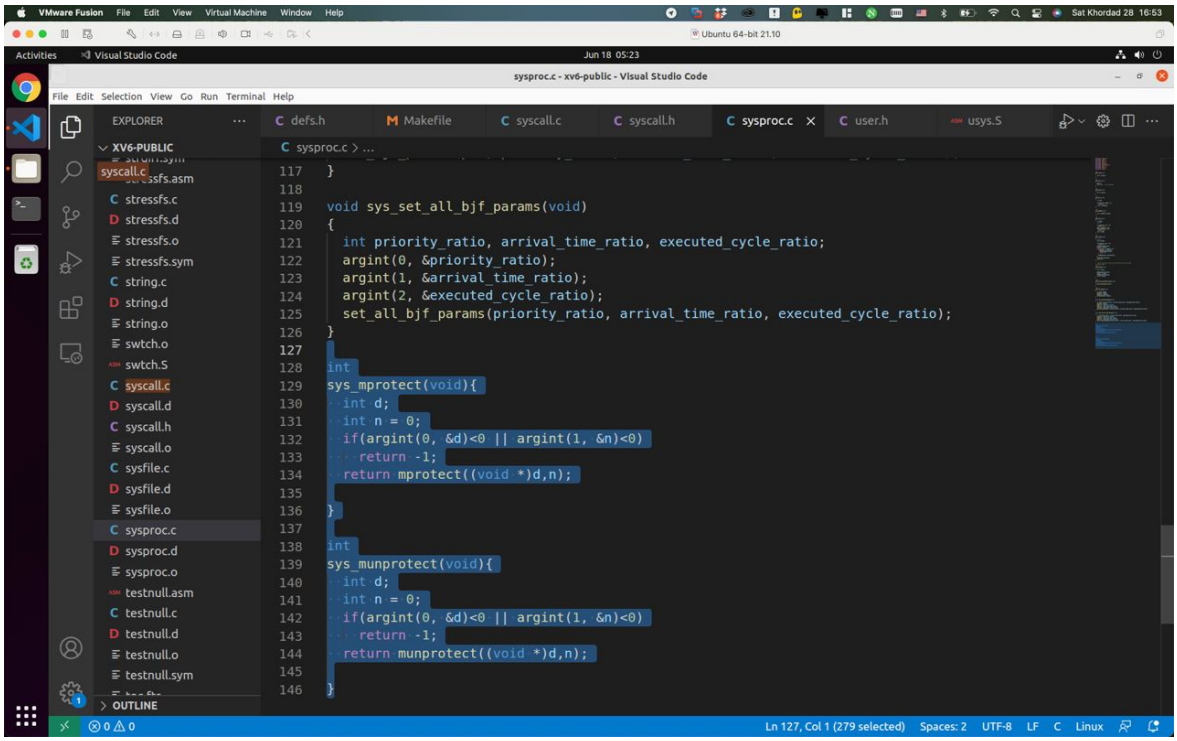

vm.c - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
Makefile
usertests.c
exec.c
syscall.c
vm.c
testnull.c
vm.c > copyuvm(pde_t*, uint)
// Given a parent process's page table, create a copy
// of it for a child.
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
pde_t *d;
pte_t *pte;
uint pa, i, flags;
char *mem;
if((d = setupkvm()) == 0)
return 0;
for(i = PGSIZE; i < sz; i += PGSIZE){
if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
panic("copyuvm: pte should exist");
if(!(*pte & PTE_P))
panic("copyuvm: page not present");
pa = PTE_ADDR(*pte);
flags = PTE_FLAGS(*pte);
if((mem = kalloc()) == 0)
goto bad;
memmove(mem, (char*)P2V(pa), PGSIZE);
if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {
kfree(mem);
goto bad;
}
}
return d;
bad:
OUTLINE
Ln 325, Col 7 (10 selected)
Spaces: 2
UTF-8
LF
C
Linux

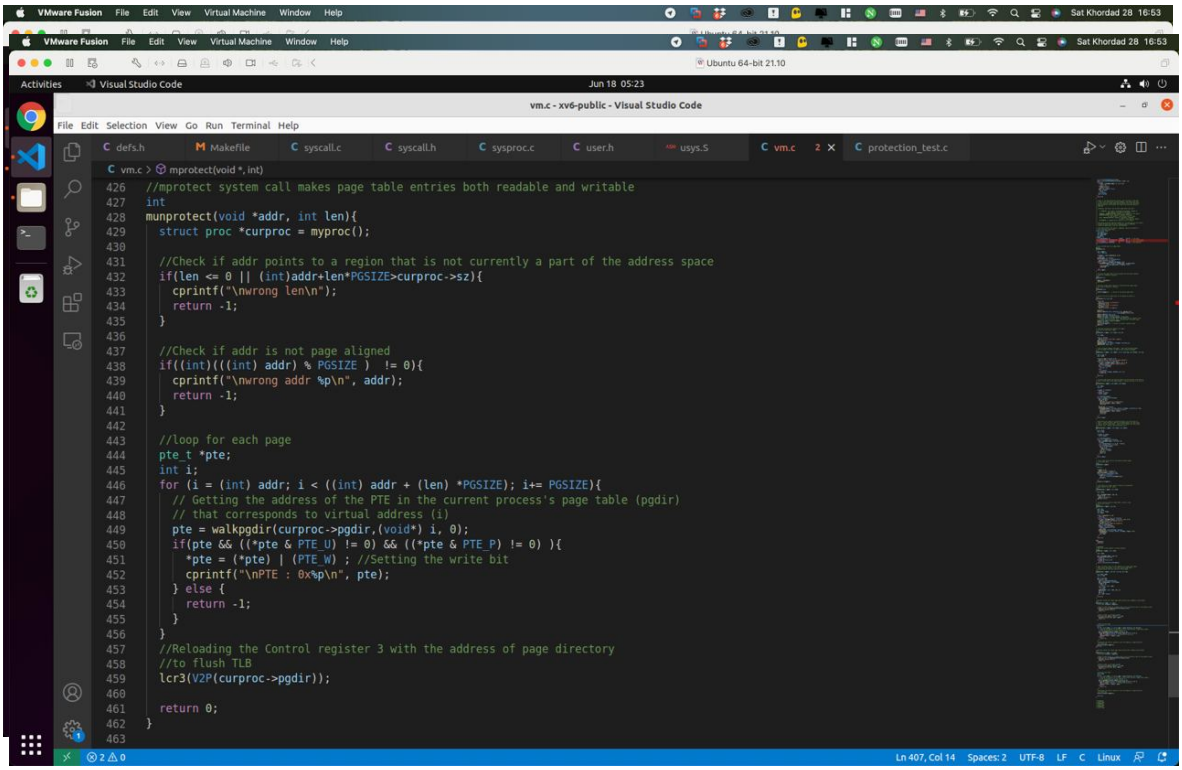

testnull.c - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
Makefile
usertests.c
exec.c
syscall.c
vm.c
testnull.c
#include "syscall.h"
#include "types.h"
#include "user.h"
#define NULL 0
#define stdout 1
int main()
{
printf(stdout, "Test program for null ptr \n");
int *p = NULL;
printf(1, "*p: %d \n",*p);
exit();
}
OUTLINE

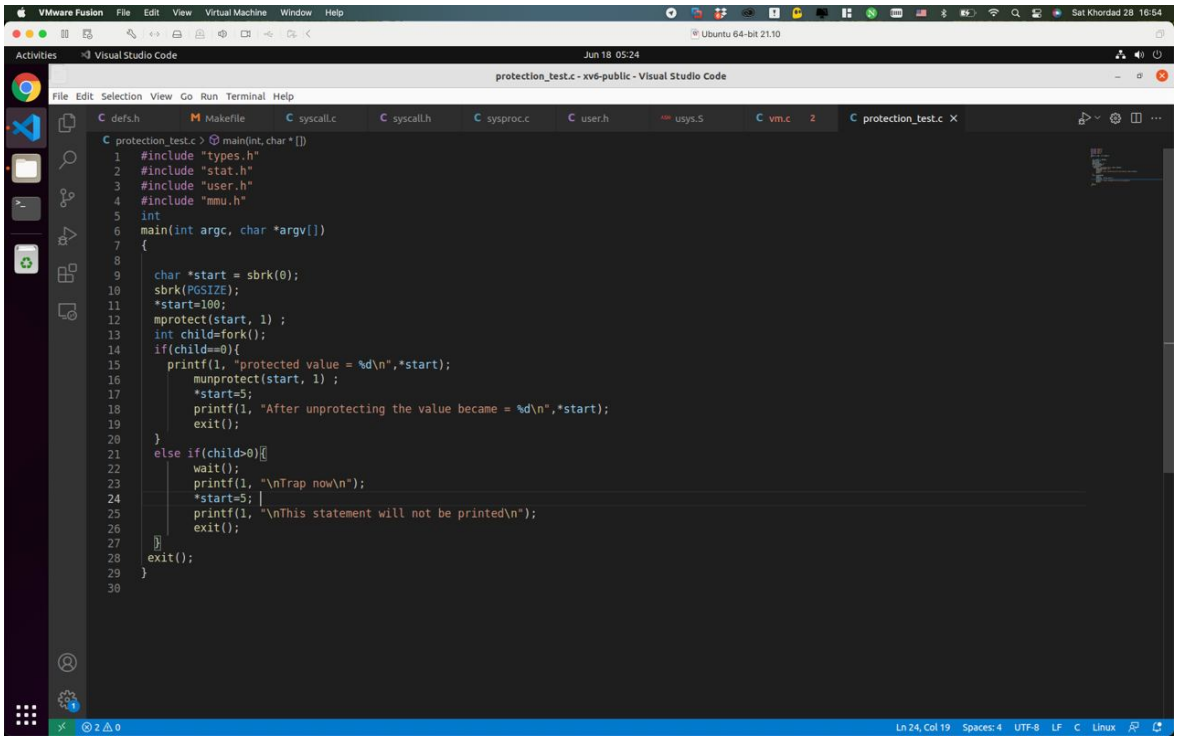

QEMU
Machine View
grep 2 6 10508
init 2 7 15176
kill 2 8 14636
ln 2 9 14532
ls 2 10 17104
mkdir 2 11 14660
rm 2 12 14640
sh 2 13 28692
stressfs 2 14 15568
usertests 2 15 62788
wc 2 16 16088
zombie 2 17 14208
strdiff 2 18 16200
ps 2 19 14184
foo 2 20 14580
set_queue 2 21 14428
set_bjf 2 22 14504
set_all_bjf 2 23 14484
testnull 2 24 14248
console 3 25 0
$ testnull
Test program for null ptr
pid 4 testnull: trap 14 err 4 on cpu 0 eip 0x101d addr 0x0--kill proc
$
testnull.c - xv6-public - Visual Studio Code
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
make
stressfs 2 14 15568
usertests 2 15 62788
wc 2 16 16088
zombie 2 17 14208
strdiff 2 18 16200
ps 2 19 14184
foo 2 20 14580
set_queue 2 21 14428
set_bjf 2 22 14504
set_all_bjf 2 23 14484
testnull 2 24 14248
console 3 25 0
$ testnull
Test program for null ptr
pid 4 testnull: trap 14 err 4 on cpu 0 eip 0x101d addr 0x0--kill proc
$
OUTLINE

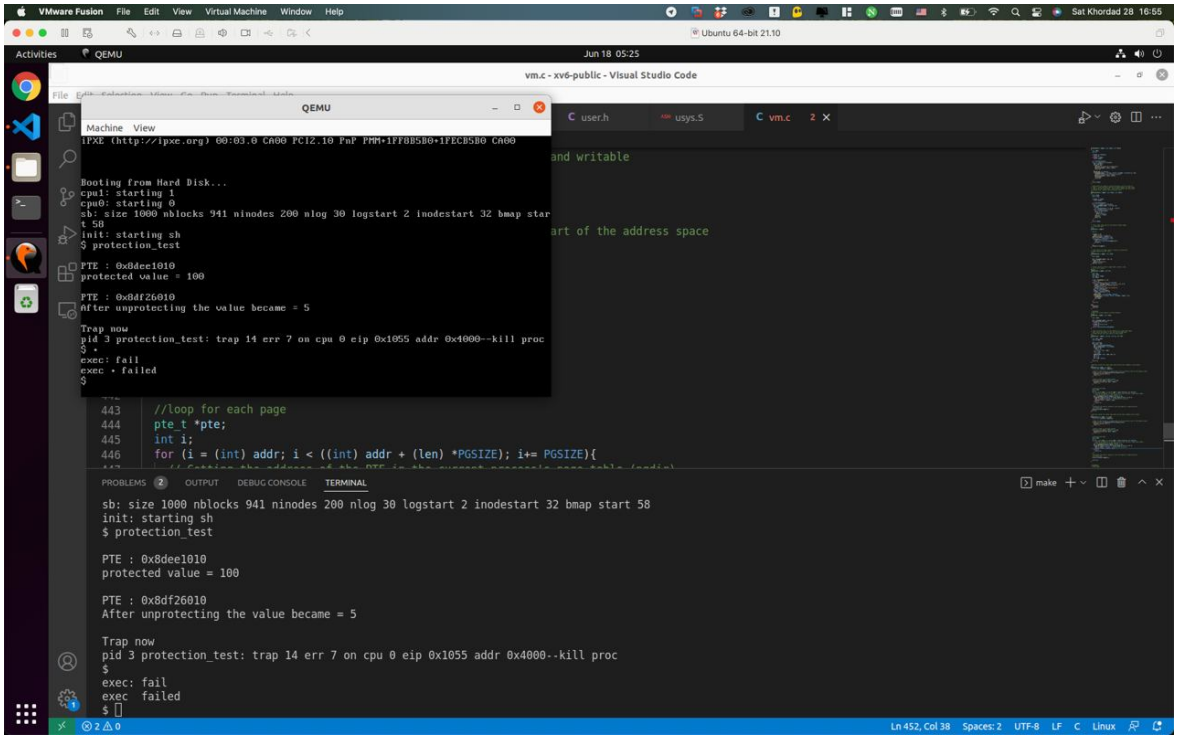

Makefile - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
Makefile
syscall
ln.sym
log.c
log.d
log.o
ls.asm
ls.c
ls.d
ls.o
ls.sym
main.c
main.d
main.o
Makefile
memide.c
memlayout.h
mkdir.asm
mkdir.c
mkdir.d
mkdir.o
mkdir.sym
mkfs
mkfs.c
mmu.h
mp.c
mp.d
OUTLINE
etags *.S *.c
vectors.S: vectors.pl
./vectors.pl > vectors.S
ULIB = ulib.o usys.o printf.o umalloc.o
_%: %.o $(ULIB)
$(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o $@ $^
$(OBJDUMP) -S $@ > $*.asm
$(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > $*.sym
_forktest: forktest.o $(ULIB)
forktest has less library code linked in - needs to be small
in order to be able to max out the proc table.
$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o _forktest forktest.o ulib.o usys.o
$(OBJDUMP) -S _forktest > forktest.asm
mkfs: mkfs.c fs.h
gcc -Werror -Wall -o mkfs mkfs.c
Prevent deletion of intermediate files, e.g. cat.o, after first build, so
that disk image changes after first build are persistent until clean. More
details:
http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o
UPROGS=\
_cat\
_echo\
Ln 147, Col 1
Tab Size: 4
UTF-8
LF
Makefile

Next Part

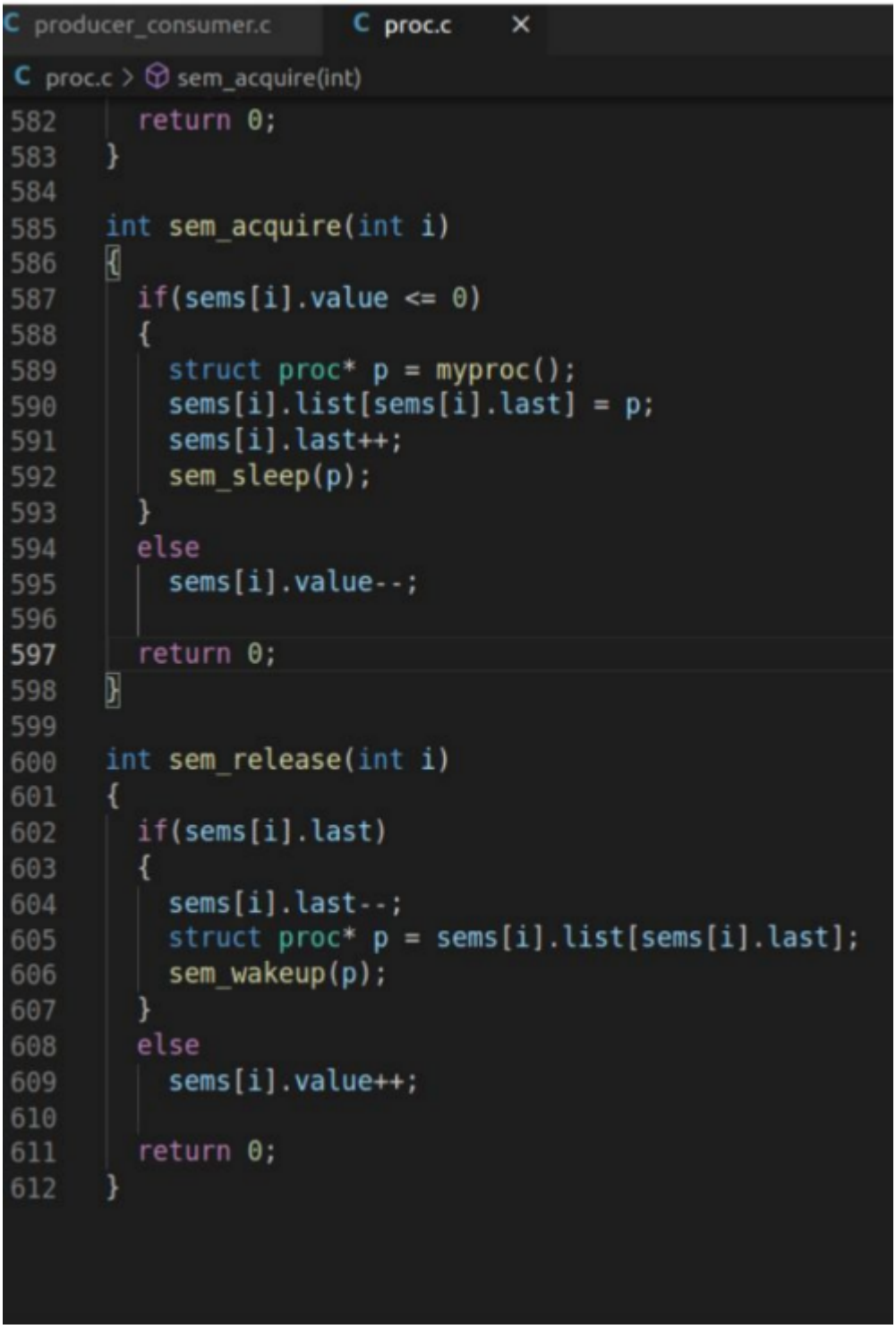

sysproc.c - xv6-public - Visual Studio Code
EXPLORER
XV6-PUBLIC
defs.h
Makefile
syscall.c
syscall.h
sysproc.c
user.h
usys.S
OUTLINE
117 }
118
119 void sys_set_all_bjf_params(void)
120 {
121 int priority_ratio, arrival_time_ratio, executed_cycle_ratio;
122 argint(0, &priority_ratio);
123 argint(1, &arrival_time_ratio);
124 argint(2, &executed_cycle_ratio);
125 set_all_bjf_params(priority_ratio, arrival_time_ratio, executed_cycle_ratio);
126 }
127
128 int
129 sys_mprotect(void){
130 int d;
131 int n = 0;
132 if(argint(0, &d)<0 || argint(1, &n)<0)
133 return -1;
134 return mprotect((void *)d,n);
135
136 }
137
138 int
139 sys_munprotect(void){
140 int d;
141 int n = 0;
142 if(argint(0, &d)<0 || argint(1, &n)<0)
143 return -1;
144 return munprotect((void *)d,n);
145
146 }


vm.c - xv6-public - Visual Studio Code
426 //mprotect system call makes page table entries both readable and writable
427 int
428 munprotect(void *addr, int len){
429 struct proc *curproc = myproc();
430
431 //Check if addr points to a region that is not currently a part of the address space
432 if(len <= 0 || (int)addr+len*PGSIZE>curproc->sz){
433 cprintf("\nwrong len\n");
434 return -1;
435 }
436
437 //Check if addr is not page aligned
438 if((int)(((int) addr) % PGSIZE) != 0){
439 cprintf("\nwrong addr %p\n", addr);
440 return -1;
441 }
442
443 //loop for each page
444 pte_t *pte;
445 int i;
446 for (i = (int) addr; i < ((int) addr + (len) *PGSIZE); i+= PGSIZE){
447 // Getting the address of the PTE in the current process's page table (pgdir)
448 // that corresponds to virtual address (i)
449 pte = walkpgdir(curproc->pgdir,(void*) i, 0);
450 if(pte && ((*pte & PTE_U) != 0) && ((*pte & PTE_P) != 0)){
451 *pte = (*pte) | (PTE_W) ; //Setting the write bit
452 cprintf("\nPTE : 0x%p\n", pte);
453 } else {
454 return -1;
455 }
456 }
457 //Reloading the Control register 3 with the address of page directory
458 //to flush TLB
459 lcr3(V2P(curproc->pgdir));
460
461 return 0;
462 }
463


protection_test.c - xv6-public - Visual Studio Code
protection_test.c > main(int, char * [])
#include "types.h"
#include "stat.h"
#include "user.h"
#include "mmu.h"
int
main(int argc, char *argv[])
{
char *start = sbrk(0);
sbrk(PGSIZE);
*start=100;
mprotect(start, 1) ;
int child=fork();
if(child==0){
printf(1, "protected value = %d\n",*start);
munprotect(start, 1) ;
*start=5;
printf(1, "After unprotecting the value became = %d\n",*start);
exit();
}
else if(child>0){
wait();
printf(1, "\nTrap now\n");
*start=5;
printf(1, "\nThis statement will not be printed\n");
exit();
}
exit();
}




```c
C producer_consumer.c    C proc.c
C proc.c > sem_release(int)
550     }
551
552     #define MAXPROC 100
553
554     typedef struct
555     {
556         int value;
557         struct proc* list[MAXPROC];
558         int last;
559     }semaphore;
560
561     semaphore sems[6];
562
563     void sem_sleep(struct proc *p1)
564     {
565       acquire(&ptable.lock);
566       p1->state = SLEEPING;
567       sched();
568       release(&ptable.lock);
569     }
570
571     void sem_wakeup(struct proc *p1)
572     {
573       acquire(&ptable.lock);
574       p1->state = RUNNABLE;
575       release(&ptable.lock);
576     }
577
578     int sem_init(int i , int v)
579     {
580       sems[i].value = v;
581       sems[i].last = 0;
582       return 0;
583     }
```


producer_consumer.c
proc.c
proc.c > sem_acquire(int)
582 return 0;
583 }
584
585 int sem_acquire(int i)
586 {
587 if(sems[i].value <= 0)
588 {
589 struct proc* p = myproc();
590 sems[i].list[sems[i].last] = p;
591 sems[i].last++;
592 sem_sleep(p);
593 }
594 else
595 sems[i].value--;
596
597 return 0;
598 }
599
600 int sem_release(int i)
601 {
602 if(sems[i].last)
603 {
604 sems[i].last--;
605 struct proc* p = sems[i].list[sems[i].last];
606 sem_wakeup(p);
607 }
608 else
609 sems[i].value++;
610
611 return 0;
612 }