Noureddine Ouelhaci

CIS-450

January 18, 2023

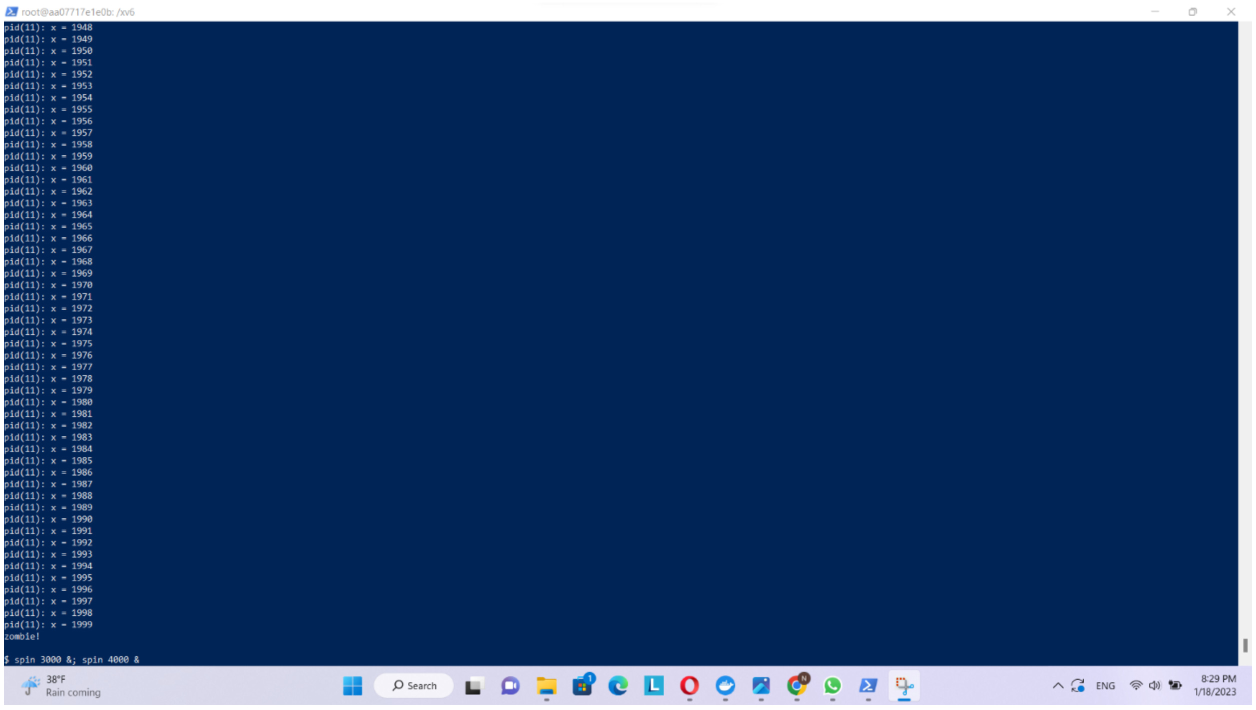
Programming 1 (P2): Building xv6

Instructor: Dr. Shengquan Wang

Part 2 Due Time: 10PM, 1/22/2023

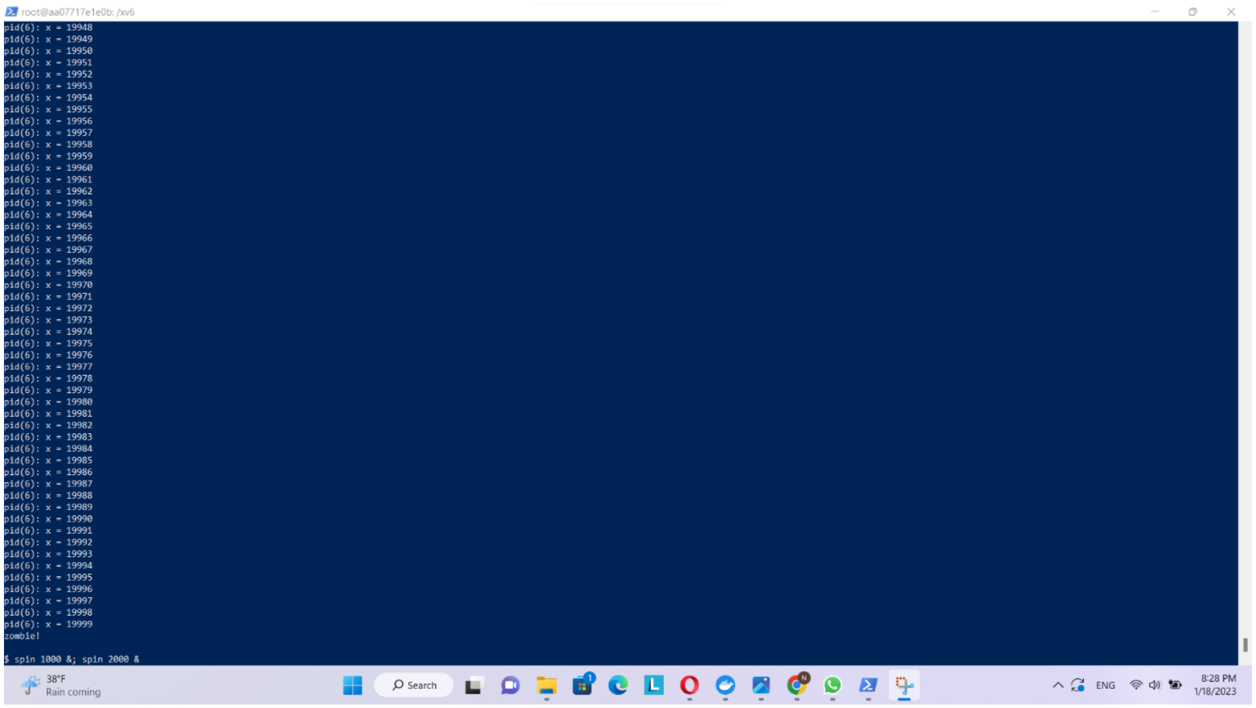
Submission:

2.1 Adding a new user program:

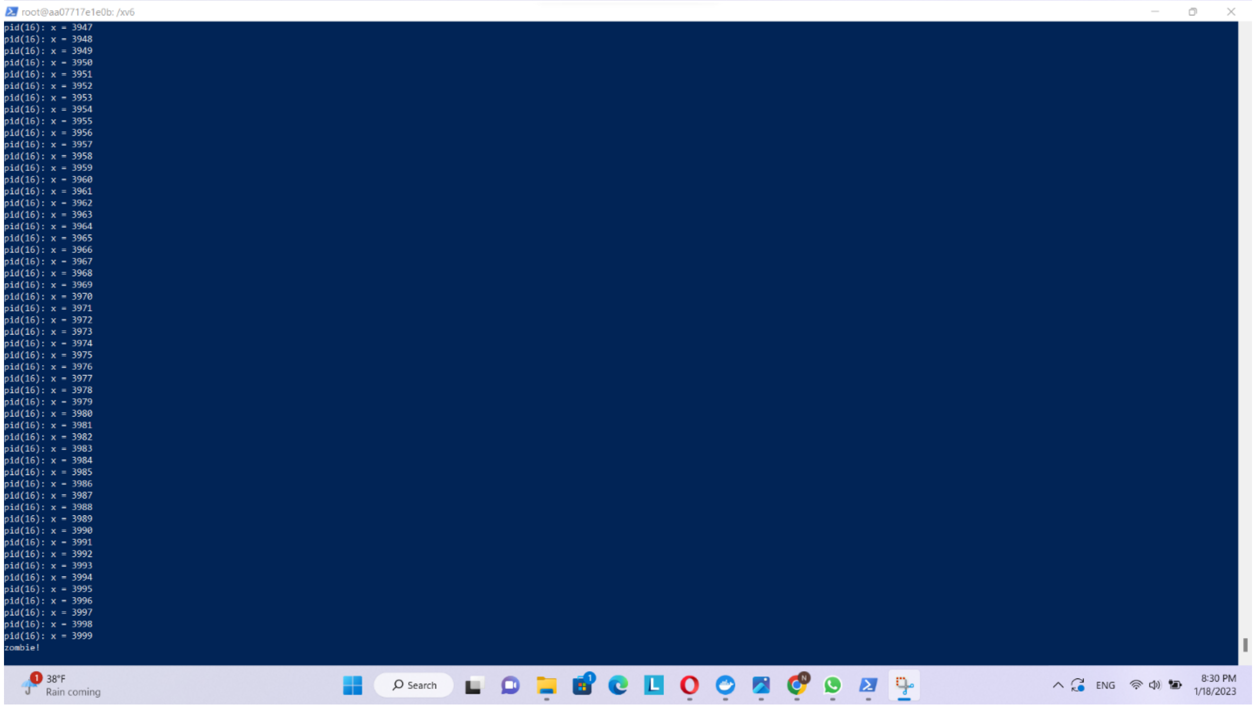


root@aa07717e1e0b: /xv6

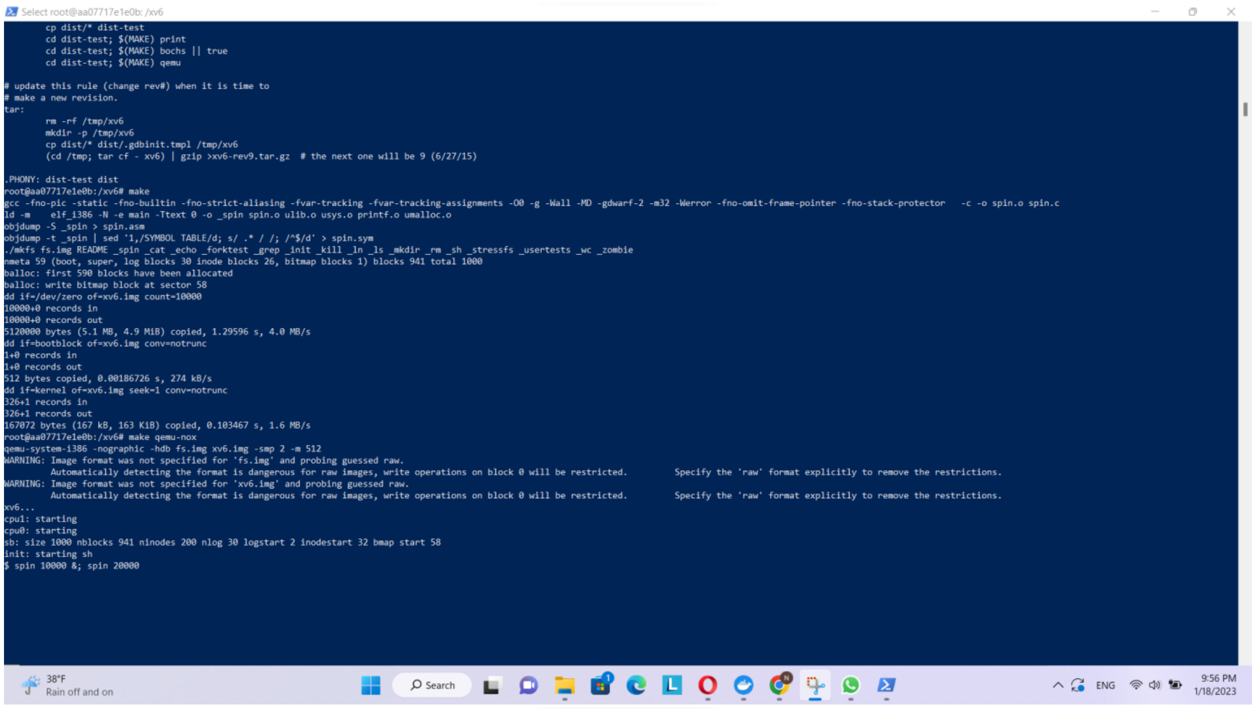
```
pid(11): x = 1948
pid(11): x = 1949
pid(11): x = 1950
pid(11): x = 1951
pid(11): x = 1952
pid(11): x = 1953
pid(11): x = 1954
pid(11): x = 1955
pid(11): x = 1956
pid(11): x = 1957
pid(11): x = 1958
pid(11): x = 1959
pid(11): x = 1960
pid(11): x = 1961
pid(11): x = 1962
pid(11): x = 1963
pid(11): x = 1964
pid(11): x = 1965
pid(11): x = 1966
pid(11): x = 1967
pid(11): x = 1968
pid(11): x = 1969
pid(11): x = 1970
pid(11): x = 1971
pid(11): x = 1972
pid(11): x = 1973
pid(11): x = 1974
pid(11): x = 1975
pid(11): x = 1976
pid(11): x = 1977
pid(11): x = 1978
pid(11): x = 1979
pid(11): x = 1980
pid(11): x = 1981
pid(11): x = 1982
pid(11): x = 1983
pid(11): x = 1984
pid(11): x = 1985
pid(11): x = 1986
pid(11): x = 1987
pid(11): x = 1988
pid(11): x = 1989
pid(11): x = 1990
pid(11): x = 1991
pid(11): x = 1992
pid(11): x = 1993
pid(11): x = 1994
pid(11): x = 1995
pid(11): x = 1996
pid(11): x = 1997
pid(11): x = 1998
pid(11): x = 1999
zombie!

$ spin 3000 &; spin 4000 &
```



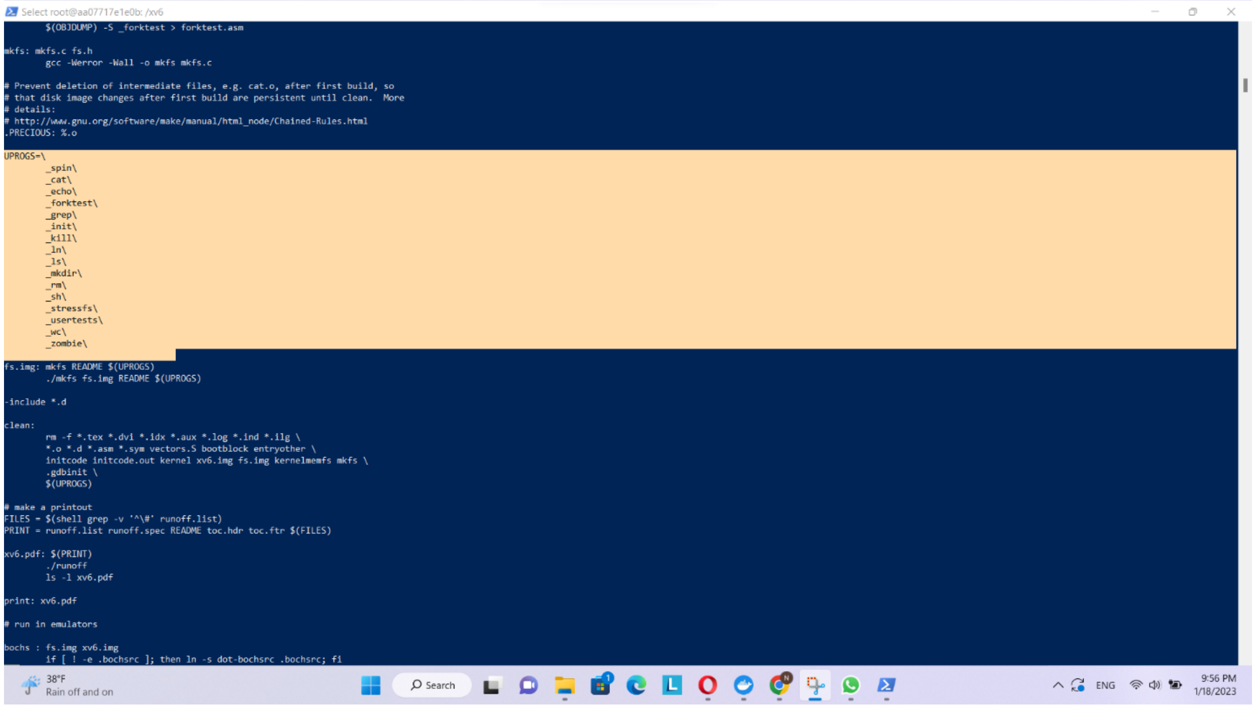
```
pid(6): x = 19948
pid(6): x = 19949
pid(6): x = 19950
pid(6): x = 19951
pid(6): x = 19952
pid(6): x = 19953
pid(6): x = 19954
pid(6): x = 19955
pid(6): x = 19956
pid(6): x = 19957
pid(6): x = 19958
pid(6): x = 19959
pid(6): x = 19960
pid(6): x = 19961
pid(6): x = 19962
pid(6): x = 19963
pid(6): x = 19964
pid(6): x = 19965
pid(6): x = 19966
pid(6): x = 19967
pid(6): x = 19968
pid(6): x = 19969
pid(6): x = 19970
pid(6): x = 19971
pid(6): x = 19972
pid(6): x = 19973
pid(6): x = 19974
pid(6): x = 19975
pid(6): x = 19976
pid(6): x = 19977
pid(6): x = 19978
pid(6): x = 19979
pid(6): x = 19980
pid(6): x = 19981
pid(6): x = 19982
pid(6): x = 19983
pid(6): x = 19984
pid(6): x = 19985
pid(6): x = 19986
pid(6): x = 19987
pid(6): x = 19988
pid(6): x = 19989
pid(6): x = 19990
pid(6): x = 19991
pid(6): x = 19992
pid(6): x = 19993
pid(6): x = 19994
pid(6): x = 19995
pid(6): x = 19996
pid(6): x = 19997
pid(6): x = 19998
pid(6): x = 19999
zombie!

$ spin 1000 &; spin 2000 &
```



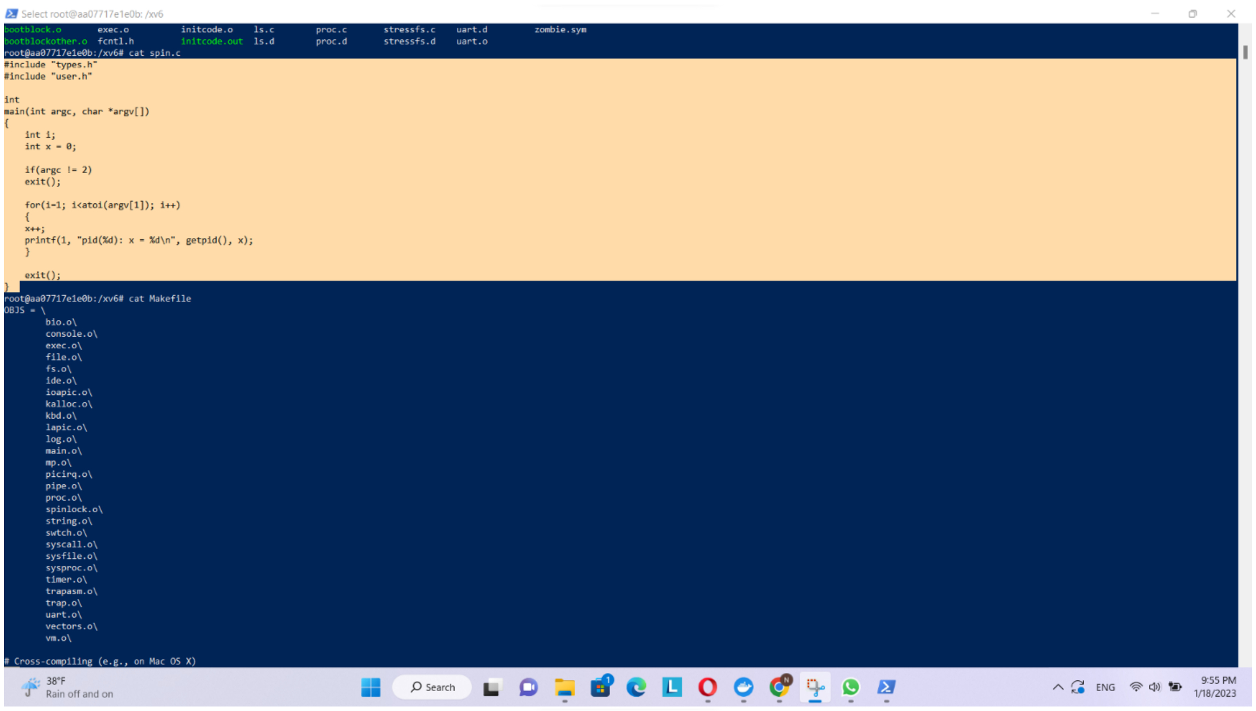
```
pid(16): x = 3947
pid(16): x = 3948
pid(16): x = 3949
pid(16): x = 3950
pid(16): x = 3951
pid(16): x = 3952
pid(16): x = 3953
pid(16): x = 3954
pid(16): x = 3955
pid(16): x = 3956
pid(16): x = 3957
pid(16): x = 3958
pid(16): x = 3959
pid(16): x = 3960
pid(16): x = 3961
pid(16): x = 3962
pid(16): x = 3963
pid(16): x = 3964
pid(16): x = 3965
pid(16): x = 3966
pid(16): x = 3967
pid(16): x = 3968
pid(16): x = 3969
pid(16): x = 3970
pid(16): x = 3971
pid(16): x = 3972
pid(16): x = 3973
pid(16): x = 3974
pid(16): x = 3975
pid(16): x = 3976
pid(16): x = 3977
pid(16): x = 3978
pid(16): x = 3979
pid(16): x = 3980
pid(16): x = 3981
pid(16): x = 3982
pid(16): x = 3983
pid(16): x = 3984
pid(16): x = 3985
pid(16): x = 3986
pid(16): x = 3987
pid(16): x = 3988
pid(16): x = 3989
pid(16): x = 3990
pid(16): x = 3991
pid(16): x = 3992
pid(16): x = 3993
pid(16): x = 3994
pid(16): x = 3995
pid(16): x = 3996
pid(16): x = 3997
pid(16): x = 3998
pid(16): x = 3999
zombie!
```



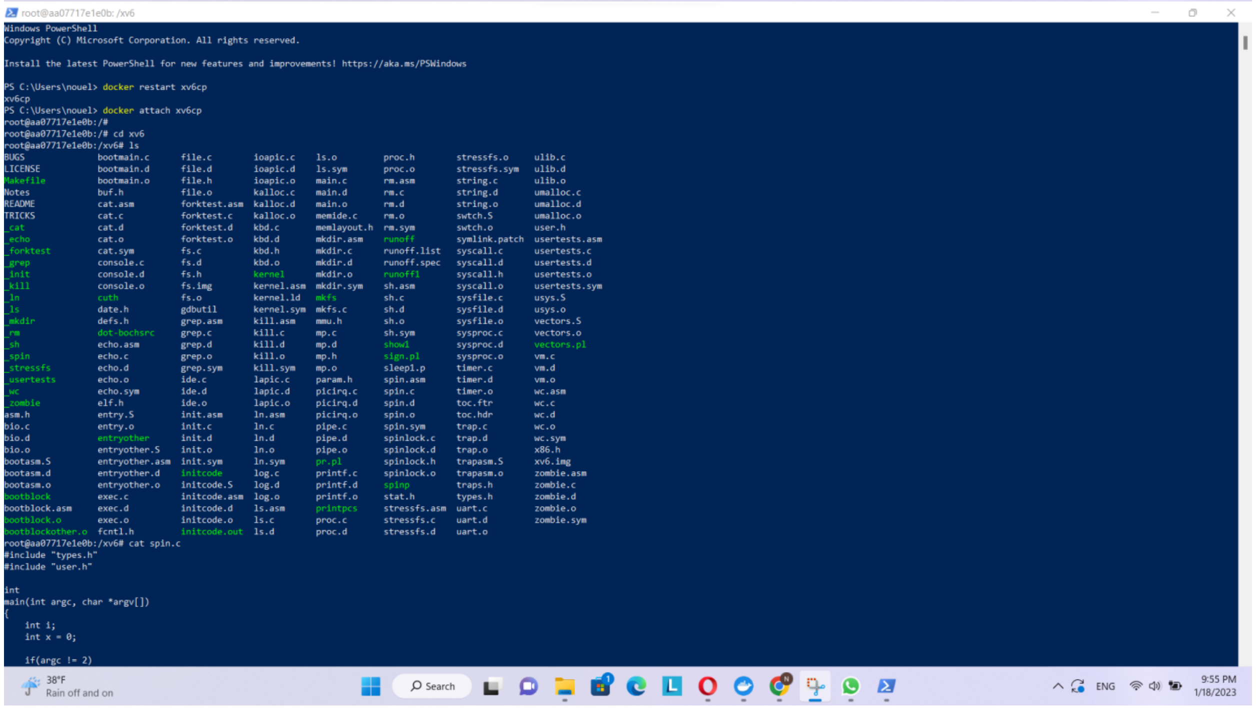
```
        cp dist/* dist-test
        cd dist-test; $(MAKE) print
        cd dist-test; $(MAKE) bochs || true
        cd dist-test; $(MAKE) qemu

# update this rule (change rev#) when it is time to
# make a new revision.
tar:
        rm -rf /tmp/xv6
        mkdir -p /tmp/xv6
        cp dist/* dist/.gdbinit.tmpl /tmp/xv6
        (cd /tmp; tar cf - xv6) | gzip >xv6-rev9.tar.gz  # the next one will be 9 (6/27/15)

.PHONY: dist-test dist
root@aa07717e1e0b:/xv6# make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -fvar-tracking -fvar-tracking-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector   -c -o spin.o spin.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _spin spin.o ulib.o usys.o printf.o umalloc.o
objdump -S _spin > spin.asm
objdump -t _spin | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > spin.sym
./mkfs fs.img README _spin _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 590 blocks have been allocated
balloc: write bitmap block at sector 58
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 1.29596 s, 4.0 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00186726 s, 274 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
326+1 records in
326+1 records out
167072 bytes (167 kB, 163 KiB) copied, 0.103467 s, 1.6 MB/s
root@aa07717e1e0b:/xv6# make qemu-nox
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ spin 10000 &; spin 20000
```



```
        $(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=\
        _spin\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\

fs.img: mkfs README $(UPROGS)
        ./mkfs fs.img README $(UPROGS)

-include *.d

clean:
        rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
        *.o *.d *.asm *.sym vectors.S bootblock entryother \
        initcode initcode.out kernel xv6.img fs.img kernelmemfs mkfs \
        .gdbinit \
        $(UPROGS)

# make a printout
FILES = $(shell grep -v '^\#' runoff.list)
PRINT = runoff.list runoff.spec README toc.hdr toc.ftr $(FILES)

xv6.pdf: $(PRINT)
        ./runoff
        ls -l xv6.pdf

print: xv6.pdf

# run in emulators

bochs : fs.img xv6.img
        if [ ! -e .bochsrc ]; then ln -s dot-bochsrc .bochsrc; fi
```

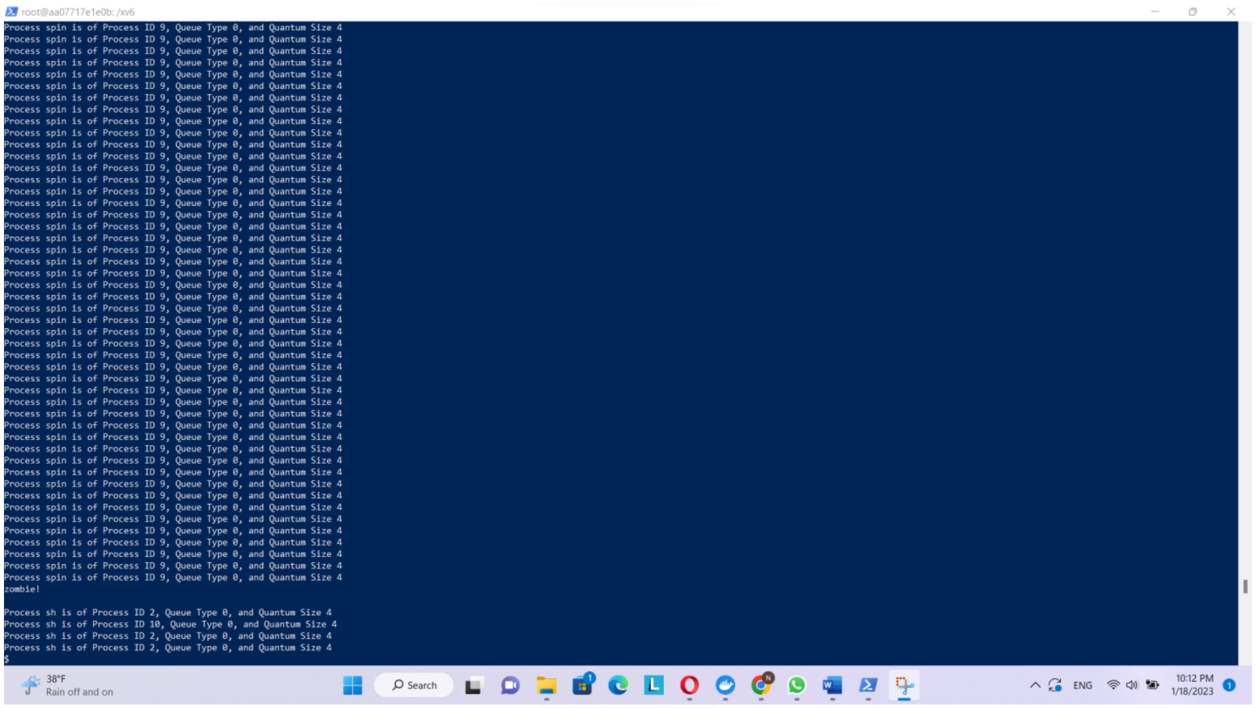


```
bootblock.o       exec.o          initcode.o    ls.c        proc.c        stressfs.c    uart.d        zombie.sym
bootblockother.o  fcntl.h         initcode.out  ls.d        proc.d        stressfs.d    uart.o
root@aa07717e1e0b:/xv6# cat spin.c
#include "types.h"
#include "user.h"

int
main(int argc, char *argv[])
{
    int i;
    int x = 0;

    if(argc != 2)
    exit();

    for(i=1; i<atoi(argv[1]); i++)
    {
    x++;
    printf(1, "pid(%d): x = %d\n", getpid(), x);
    }

    exit();
}
root@aa07717e1e0b:/xv6# cat Makefile
OBJS = \
        bio.o\
        console.o\
        exec.o\
        file.o\
        fs.o\
        ide.o\
        ioapic.o\
        kalloc.o\
        kbd.o\
        lapic.o\
        log.o\
        main.o\
        mp.o\
        picirq.o\
        pipe.o\
        proc.o\
        spinlock.o\
        string.o\
        swtch.o\
        syscall.o\
        sysfile.o\
        sysproc.o\
        timer.o\
        trapasm.o\
        trap.o\
        uart.o\
        vectors.o\
        vm.o\

# Cross-compiling (e.g., on Mac OS X)
```

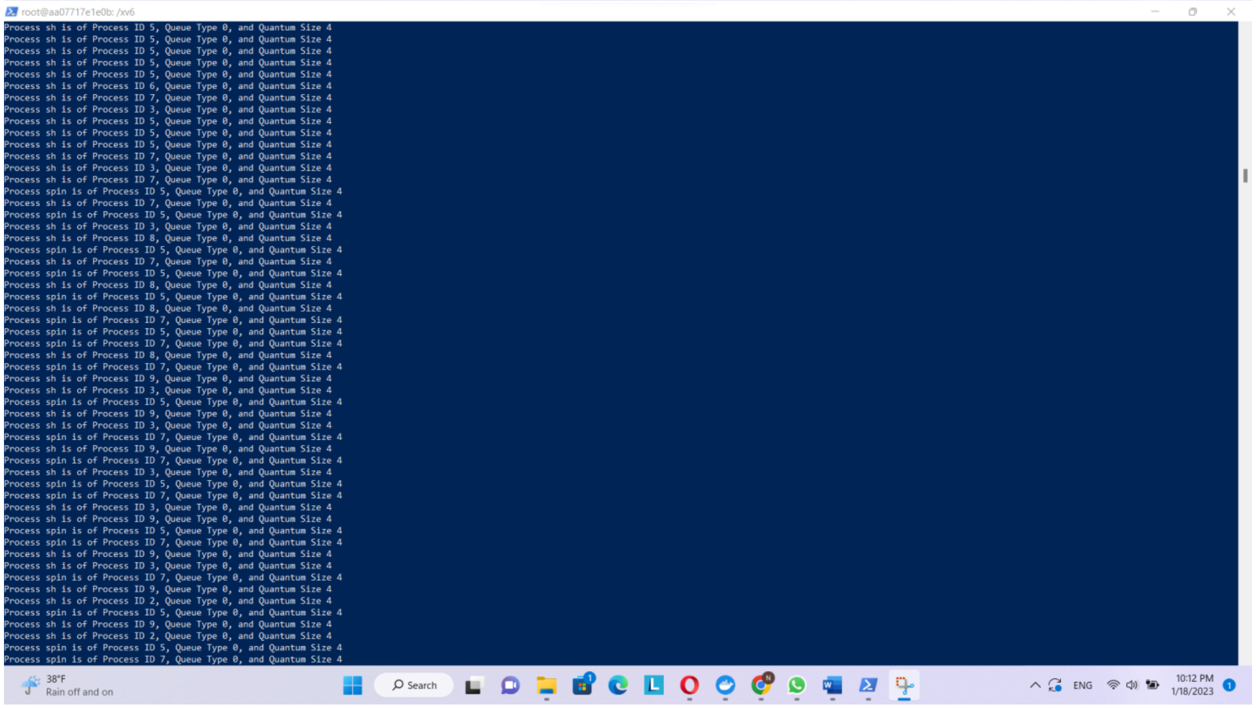


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\nouel> docker restart xv6cp
xv6cp
PS C:\Users\nouel> docker attach xv6cp
root@aa07717e1e0b:/#
root@aa07717e1e0b:/# cd xv6
root@aa07717e1e0b:/xv6# ls
BUGS              bootmain.c      file.c          ioapic.c    ls.o          proc.h         stressfs.o     ulib.c
LICENSE           bootmain.d      file.d          ioapic.d    ls.sym        proc.o         stressfs.sym   ulib.d
Makefile          bootmain.o      file.h          ioapic.o    main.c        rm.asm         string.c       ulib.o
Notes             buf.h           file.o          kalloc.c    main.d        rm.c           string.d       umalloc.c
README            cat.asm         forktest.asm    kalloc.d    main.o        rm.d           string.o       umalloc.d
TRICKS            cat.c           forktest.c      kalloc.o    memide.c      rm.o           swtch.S        umalloc.o
_cat              cat.d           forktest.d      kbd.c       memlayout.h   rm.sym         swtch.o        user.h
_echo             cat.o           forktest.o      kbd.d       mkdir.asm     runoff         symlink.patch  usertests.asm
_forktest         cat.sym         fs.c            kbd.h       mkdir.c       runoff.list    syscall.c      usertests.c
_grep             console.c       fs.d            kbd.o       mkdir.d       runoff.spec    syscall.d      usertests.d
_init             console.d       fs.h            kernel      mkdir.o       runoff1        syscall.h      usertests.o
_kill             console.o       fs.img          kernel.asm  mkdir.sym     sh.asm         syscall.o      usertests.sym
_ln               cuth            fs.o            kernel.ld   mkfs          sh.c           sysfile.c      usys.S
_ls               date.h          gdbutil         kernel.sym  mkfs.c        sh.d           sysfile.d      usys.o
_mkdir            defs.h          grep.asm        kill.asm    mmu.h         sh.o           sysfile.o      vectors.S
_rm               dot-bochsrc     grep.c          kill.c      mp.c          sh.sym         sysproc.c      vectors.o
_sh               echo.asm        grep.d          kill.d      mp.d          show1          sysproc.d      vectors.pl
_spin             echo.c          grep.o          kill.o      mp.h          sign.pl        sysproc.o      vm.c
_stressfs         echo.d          grep.sym        kill.sym    mp.o          sleep1.p       timer.c        vm.d
_usertests        echo.o          ide.c           lapic.c     param.h       spin.asm       timer.d        vm.o
_wc               echo.sym        ide.d           lapic.d     picirq.c      spin.c         timer.o        wc.asm
_zombie           elf.h           ide.o           lapic.o     picirq.d      spin.d         toc.ftr        wc.c
asm.h             entry.S         init.asm        ln.asm      picirq.o      spin.o         toc.hdr        wc.d
bio.c             entry.o         init.c          ln.c        pipe.c        spin.sym       trap.c         wc.o
bio.d             entryother      init.d          ln.d        pipe.d        spinlock.c     trap.d         wc.sym
bio.o             entryother.S    init.o          ln.o        pipe.o        spinlock.d     trap.o         x86.h
bootasm.S         entryother.asm  init.sym        ln.sym      pr.pl         spinlock.h     trapasm.S      xv6.img
bootasm.d         entryother.d    initcode        log.c       printf.c      spinlock.o     trapasm.o      zombie.asm
bootasm.o         entryother.o    initcode.S      log.d       printf.d      spinp          traps.h        zombie.c
bootblock         exec.c          initcode.asm    log.o       printf.o      stat.h         types.h        zombie.d
bootblock.asm     exec.d          initcode.d      ls.asm      printpcs      stressfs.asm   uart.c         zombie.o
bootblock.o       exec.o          initcode.o      ls.c        proc.c        stressfs.c     uart.d         zombie.sym
bootblockother.o  fcntl.h         initcode.out    ls.d        proc.d        stressfs.d     uart.o
root@aa07717e1e0b:/xv6# cat spin.c
#include "types.h"
#include "user.h"

int
main(int argc, char *argv[])
{
    int i;
    int x = 0;

    if(argc != 2)
```

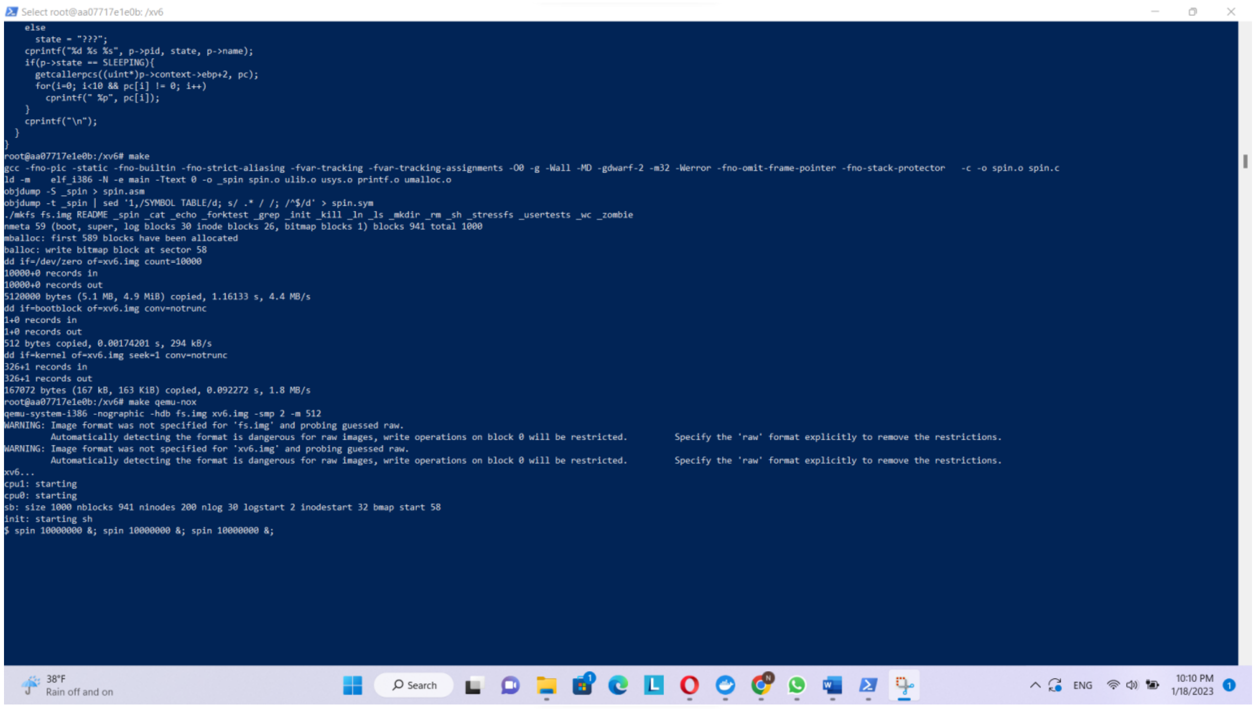
<u>2.2 Modifying and accessing PCB:</u>



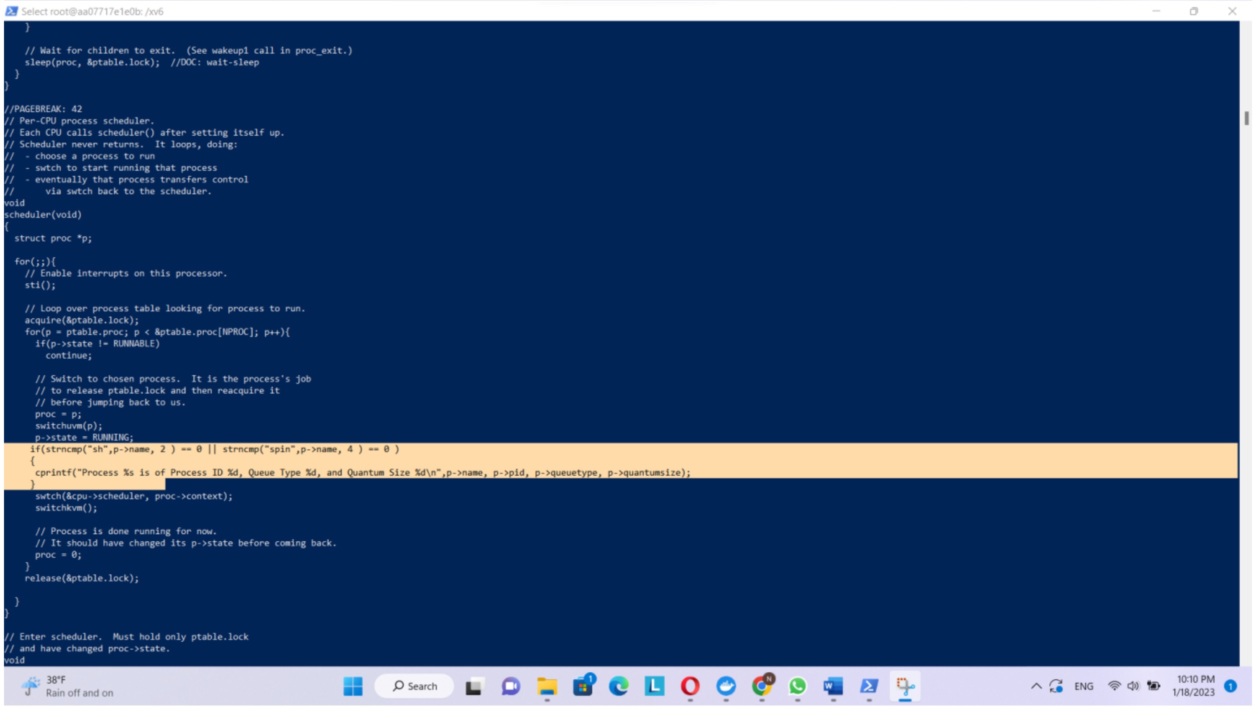
```
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 9, Queue Type 0, and Quantum Size 4
zombie!

Process sh is of Process ID 2, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 10, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 2, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 2, Queue Type 0, and Quantum Size 4
$
```



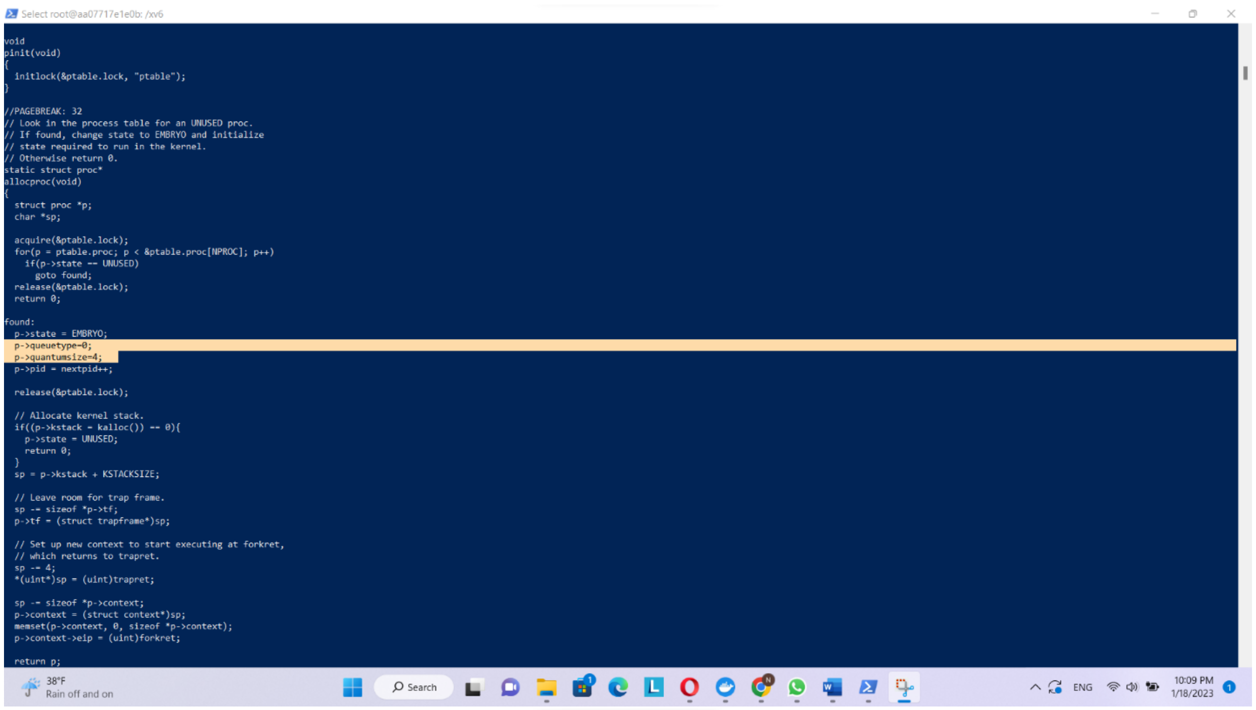
```
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 6, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 7, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 7, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 8, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 7, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 8, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 8, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 8, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 3, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 2, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 9, Queue Type 0, and Quantum Size 4
Process sh is of Process ID 2, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 5, Queue Type 0, and Quantum Size 4
Process spin is of Process ID 7, Queue Type 0, and Quantum Size 4
```


Select root@aa07717e1e0b: /xv6

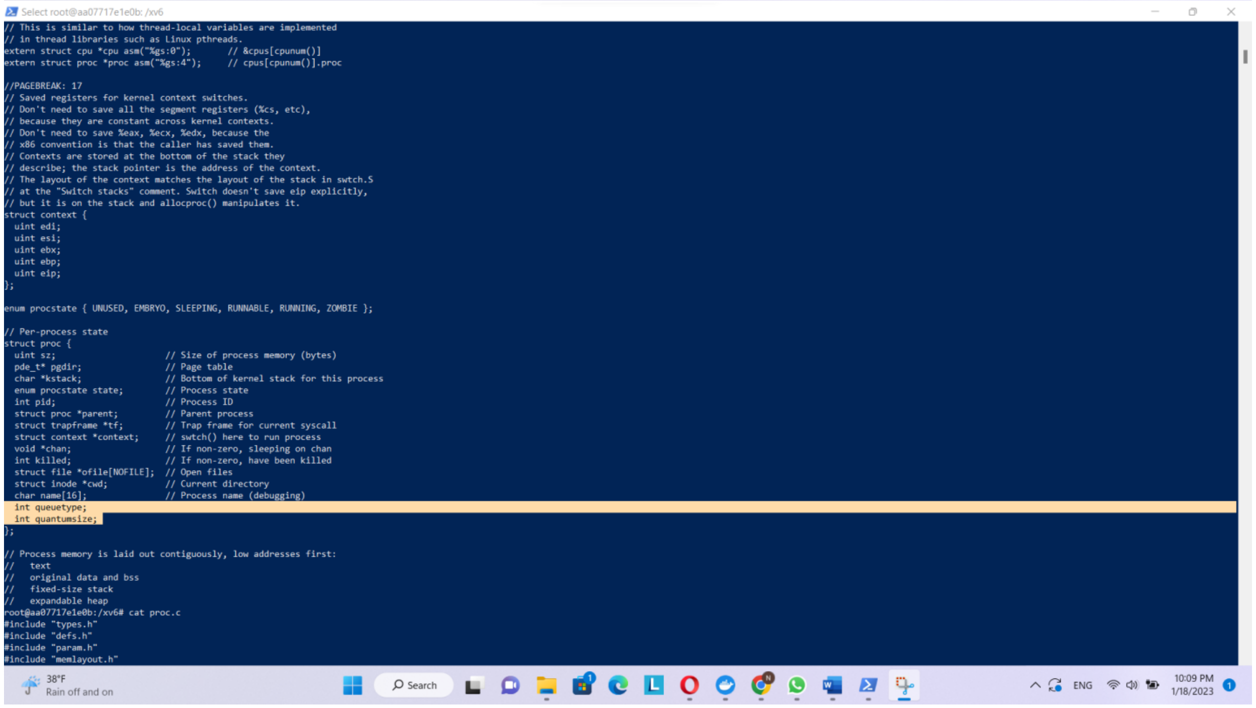
```
    else
      state = "???";
    cprintf("%d %s %s", p->pid, state, p->name);
    if(p->state == SLEEPING){
      getcallerpcs((uint*)p->context->ebp+2, pc);
      for(i=0; i<10 && pc[i] != 0; i++)
        cprintf(" %p", pc[i]);
    }
    cprintf("\n");
  }
}
root@aa07717e1e0b:/xv6# make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -fvar-tracking -fvar-tracking-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector   -c -o spin.o spin.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _spin spin.o ulib.o usys.o printf.o umalloc.o
objdump -S _spin > spin.asm
objdump -t _spin | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > spin.sym
./mkfs fs.img README _spin _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
mballoc: first 589 blocks have been allocated
balloc: write bitmap block at sector 58
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 1.16133 s, 4.4 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00174201 s, 294 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
326+1 records in
326+1 records out
167072 bytes (167 kB, 163 KiB) copied, 0.092272 s, 1.8 MB/s
root@aa07717e1e0b:/xv6# make qemu-nox
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ spin 10000000 &; spin 10000000 &; spin 10000000 &;
```


Select root@aa07717e1e0b: /xv6

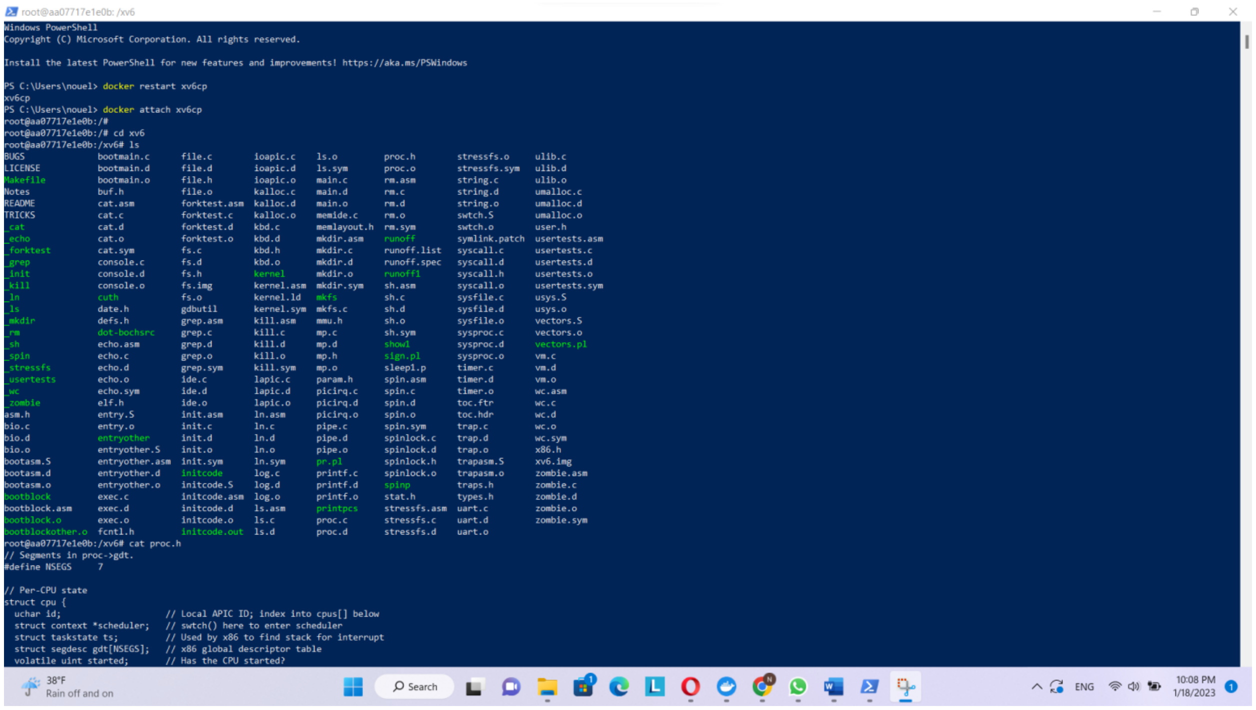
```
    }

    // Wait for children to exit.  (See wakeup1 call in proc_exit.)
    sleep(proc, &ptable.lock);  //DOC: wait-sleep
  }
}

//PAGEBREAK: 42
// Per-CPU process scheduler.
// Each CPU calls scheduler() after setting itself up.
// Scheduler never returns.  It loops, doing:
//  - choose a process to run
//  - swtch to start running that process
//  - eventually that process transfers control
//      via swtch back to the scheduler.
void
scheduler(void)
{
  struct proc *p;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
     if(strncmp("sh",p->name, 2 ) == 0 || strncmp("spin",p->name, 4 ) == 0 )
     {
      cprintf("Process %s is of Process ID %d, Queue Type %d, and Quantum Size %d\n",p->name, p->pid, p->queuetype, p->quantumsize);
     }
      swtch(&cpu->scheduler, proc->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      proc = 0;
    }
    release(&ptable.lock);

  }
}

// Enter scheduler.  Must hold only ptable.lock
// and have changed proc->state.
void
```



```
void
pinit(void)
{
  initlock(&ptable.lock, "ptable");
}

//PAGEBREAK: 32
// Look in the process table for an UNUSED proc.
// If found, change state to EMBRYO and initialize
// state required to run in the kernel.
// Otherwise return 0.
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;
  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->queuetype=0;
  p->quantumsize=4;
  p->pid = nextpid++;

  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  // Set up new context to start executing at forkret,
  // which returns to trapret.
  sp -= 4;
  *(uint*)sp = (uint)trapret;

  sp -= sizeof *p->context;
  p->context = (struct context*)sp;
  memset(p->context, 0, sizeof *p->context);
  p->context->eip = (uint)forkret;

  return p;
```



```
// This is similar to how thread-local variables are implemented
// in thread libraries such as Linux pthreads.
extern struct cpu *cpu asm("%gs:0");       // &cpus[cpunum()]
extern struct proc *proc asm("%gs:4");     // cpus[cpunum()].proc

//PAGEBREAK: 17
// Saved registers for kernel context switches.
// Don't need to save all the segment registers (%cs, etc),
// because they are constant across kernel contexts.
// Don't need to save %eax, %ecx, %edx, because the
// x86 convention is that the caller has saved them.
// Contexts are stored at the bottom of the stack they
// describe; the stack pointer is the address of the context.
// The layout of the context matches the layout of the stack in swtch.S
// at the "Switch stacks" comment. Switch doesn't save eip explicitly,
// but it is on the stack and allocproc() manipulates it.
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};

enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int queuetype;
  int quantumsize;
};

// Process memory is laid out contiguously, low addresses first:
//   text
//   original data and bss
//   fixed-size stack
//   expandable heap
root@aa07717e1e0b:/xv6# cat proc.c
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
```



root@aa07717e1e0b: /xv6

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\nouel> docker restart xv6cp
xv6cp
PS C:\Users\nouel> docker attach xv6cp
root@aa07717e1e0b:/#
root@aa07717e1e0b:/# cd xv6
root@aa07717e1e0b:/xv6# ls
BUGS              bootmain.c      file.c        ioapic.c    ls.o         proc.h        stressfs.o     ulib.c
LICENSE           bootmain.d      file.d        ioapic.d    ls.sym       proc.o        stressfs.sym   ulib.d
Makefile          bootmain.o      file.h        ioapic.o    main.c       rm.asm        string.c       ulib.o
Notes             buf.h           file.o        kalloc.c    main.d       rm.c          string.d       umalloc.c
README            cat.asm         forktest.asm  kalloc.d    main.o       rm.d          string.o       umalloc.d
TRICKS            cat.c           forktest.c    kalloc.o    memide.c     rm.o          swtch.S        umalloc.o
_cat              cat.d           forktest.d    kbd.c       memlayout.h  rm.sym        swtch.o        user.h
_echo             cat.o           forktest.o    kbd.d       mkdir.asm    runoff        symlink.patch  usertests.asm
_forktest         cat.sym         fs.c          kbd.h       mkdir.c      runoff.list   syscall.c      usertests.c
_grep             console.c       fs.d          kbd.o       mkdir.d      runoff.spec   syscall.d      usertests.d
_init             console.d       fs.h          kernel      mkdir.o      runoff1       syscall.h      usertests.o
_kill             console.o       fs.img        kernel.asm  mkdir.sym    sh.asm        syscall.o      usertests.sym
_ln               cuth            fs.o          kernel.ld   mkfs         sh.c          sysfile.c      usys.S
_ls               date.h          gdbutil       kernel.sym  mkfs.c       sh.d          sysfile.d      usys.o
_mkdir            defs.h          grep.asm      kill.asm    mmu.h        sh.o          sysfile.o      vectors.S
_rm               dot-bochsrc     grep.c        kill.c      mp.c         sh.sym        sysproc.c      vectors.o
_sh               echo.asm        grep.d        kill.d      mp.d         show1         sysproc.d      vectors.pl
_spin             echo.c          grep.o        kill.o      mp.h         sign.pl       sysproc.o      vm.c
_stressfs         echo.d          grep.sym      kill.sym    mp.o         sleep1.p      timer.c        vm.d
_usertests        echo.o          ide.c         lapic.c     param.h      spin.asm      timer.d        vm.o
_wc               echo.sym        ide.d         lapic.d     picirq.c     spin.c        timer.o        wc.asm
_zombie           elf.h           ide.o         lapic.o     picirq.d     spin.d        toc.ftr        wc.c
asm.h             entry.S         init.asm      ln.asm      picirq.o     spin.o        toc.hdr        wc.d
bio.c             entry.o         init.c        ln.c        pipe.c       spin.sym      trap.c         wc.o
bio.d             entryother      init.d        ln.d        pipe.d       spinlock.c    trap.d         wc.sym
bio.o             entryother.S    init.o        ln.o        pipe.o       spinlock.d    trap.o         x86.h
bootasm.S         entryother.asm  init.sym      ln.sym      pr.pl        spinlock.h    trapasm.S      xv6.img
bootasm.d         entryother.d    initcode      log.c       printf.c     spinlock.o    trapasm.o      zombie.asm
bootasm.o         entryother.o    initcode.S    log.d       printf.d     spinp         traps.h        zombie.c
bootblock         exec.c          initcode.asm  log.o       printf.o     stat.h        types.h        zombie.d
bootblock.asm     exec.d          initcode.d    ls.asm      printpcs     stressfs.asm  uart.c         zombie.o
bootblock.o       exec.o          initcode.o    ls.c        proc.c       stressfs.c    uart.d         zombie.sym
bootblockother.o  fcntl.h         initcode.out  ls.d        proc.d       stressfs.d    uart.o
root@aa07717e1e0b:/xv6# cat proc.h
// Segments in proc->gdt.
#define NSEGS     7

// Per-CPU state
struct cpu {
  uchar id;                    // Local APIC ID; index into cpus[] below
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
```