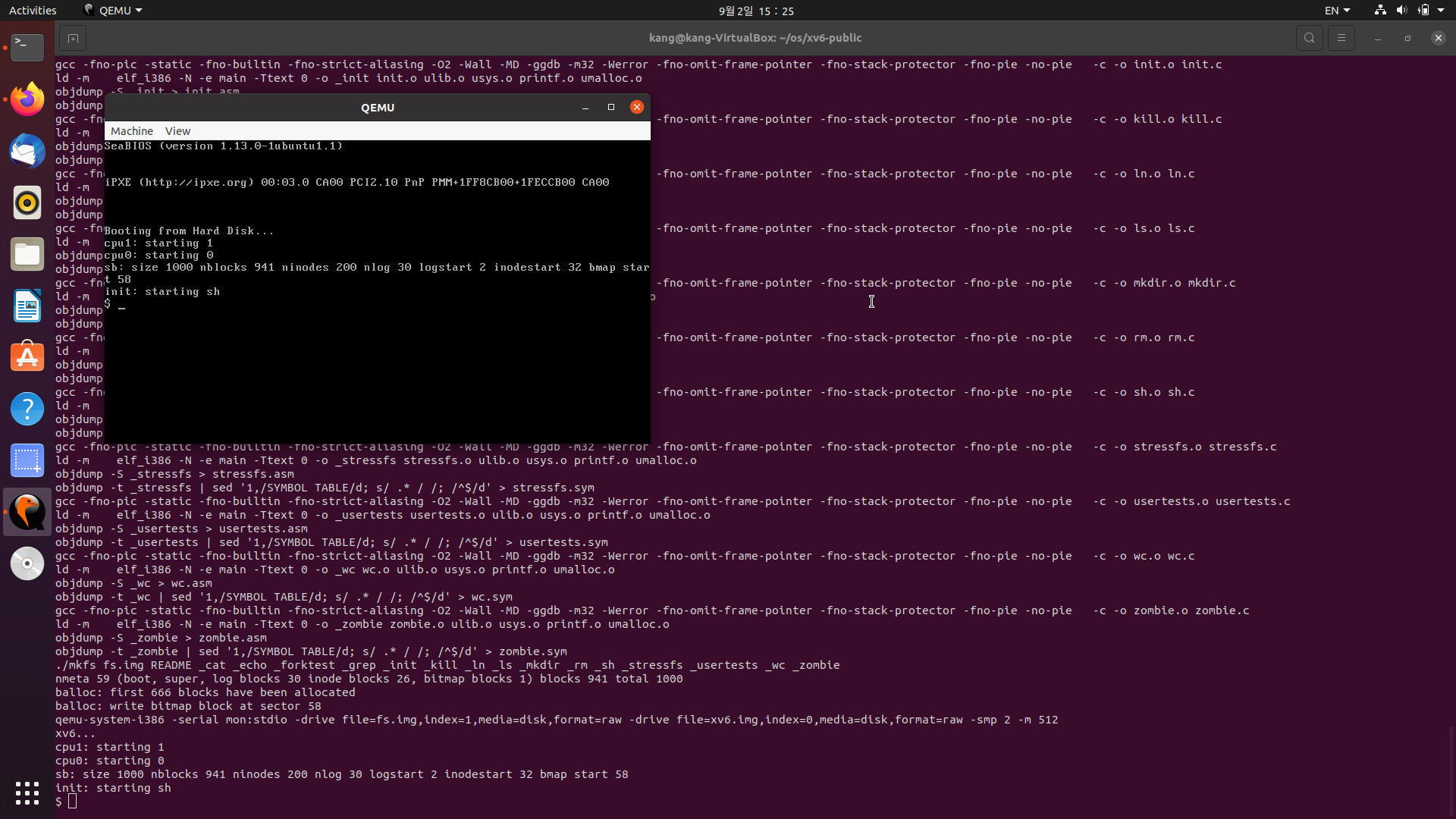
컴퓨터학부, 20202925, 강준현, kang59

**1. 개요**

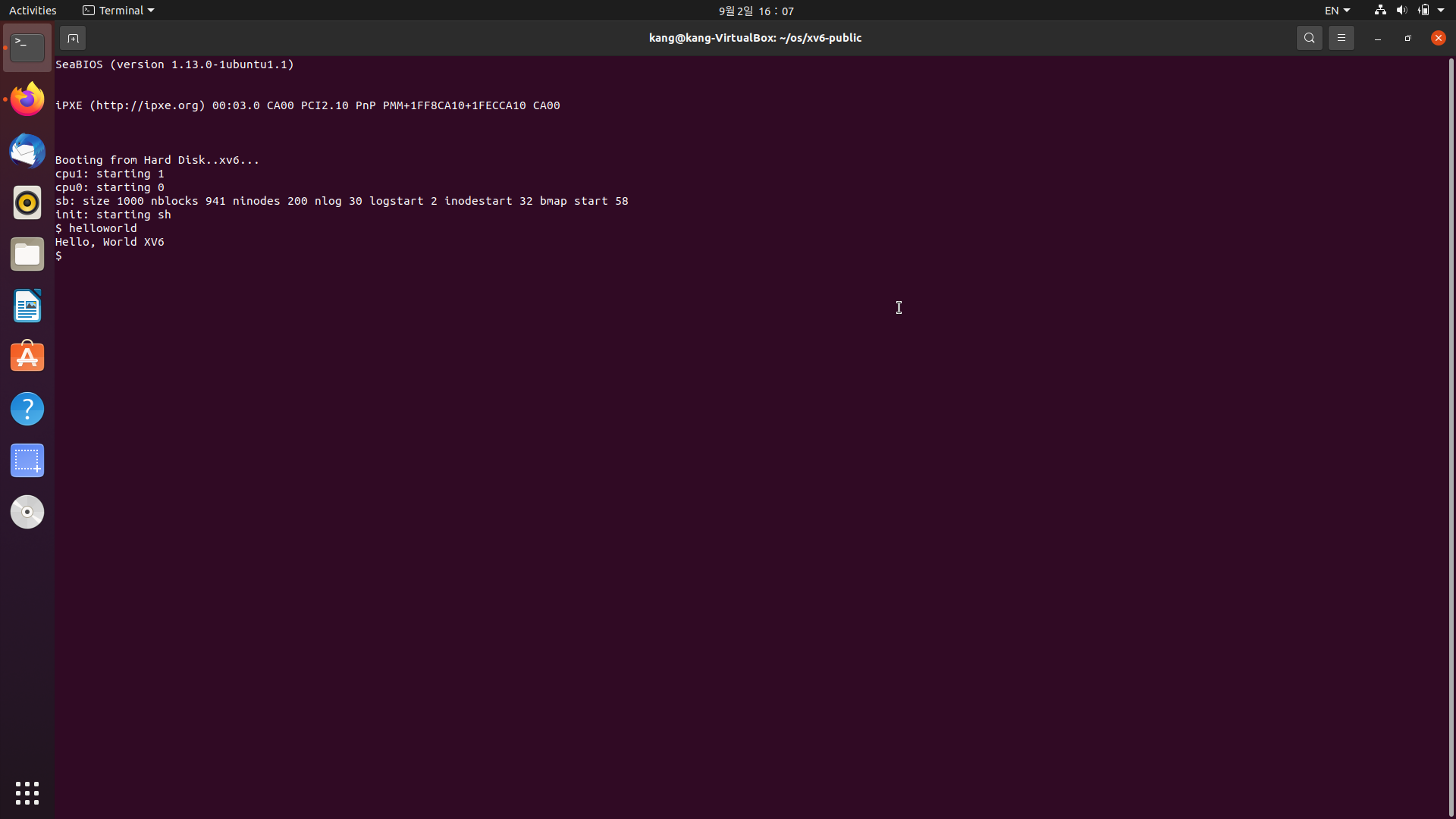
xv6설치 및 컴파일을 해본다. git clone을 통해 다운하고, qemu 에뮬레이터에서 실행본다. 간단한 쉘프로그램 작성, Makefile 수정을 통해 xv6를 파악해본다. hcat 쉘프로그램을 통해 원하는 줄 수 만큼 원하는 파일의 내용을 읽는 프로그램을 작성해본다. ssu\_login쉘프로그램을 통해 xv6시작을 로그인을 해야 시작되게 만들어본다. xv6의 흐름을 대략적을 파악해본다.

**2. 결과**

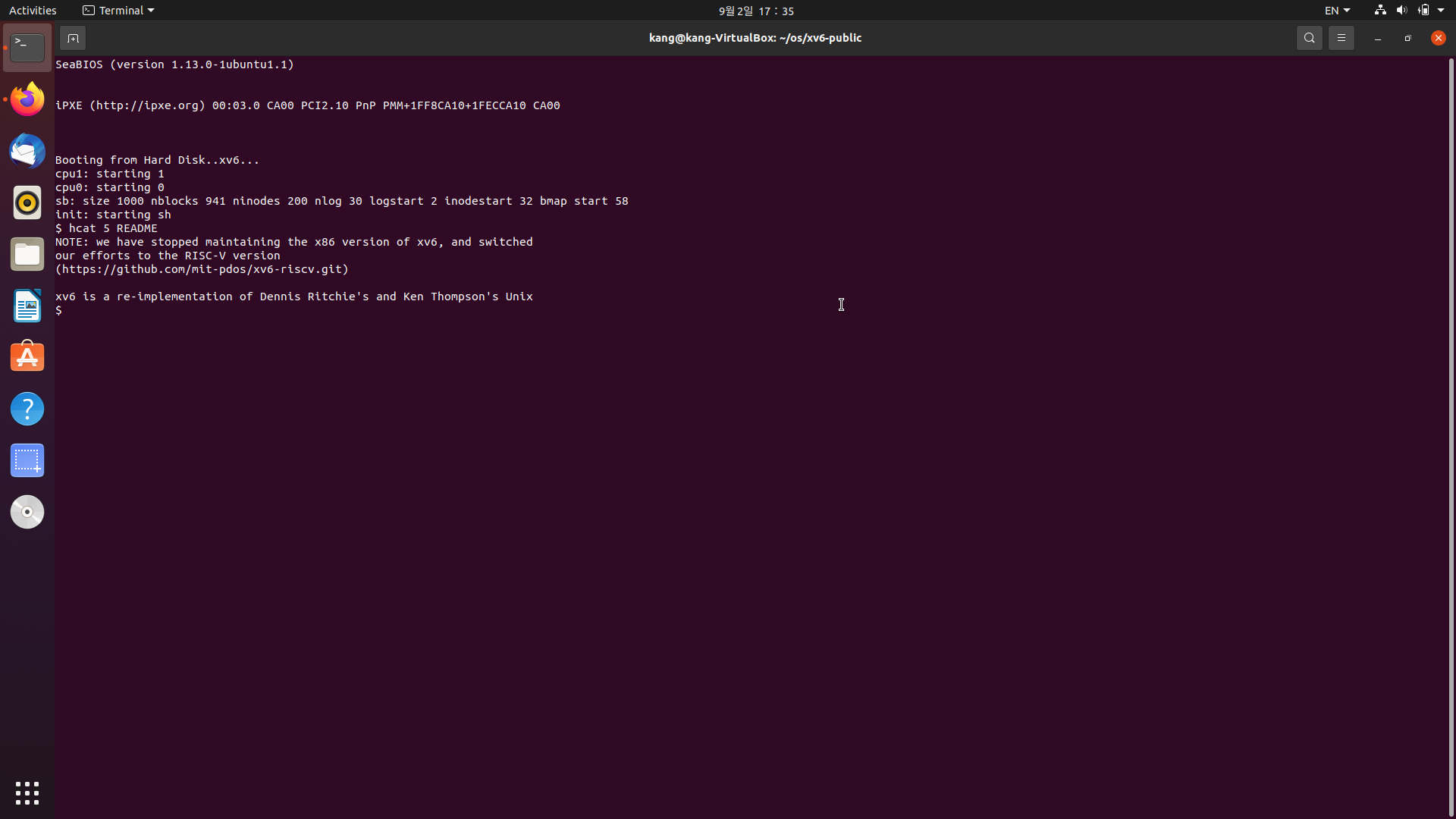
1. make qemu 로 실행

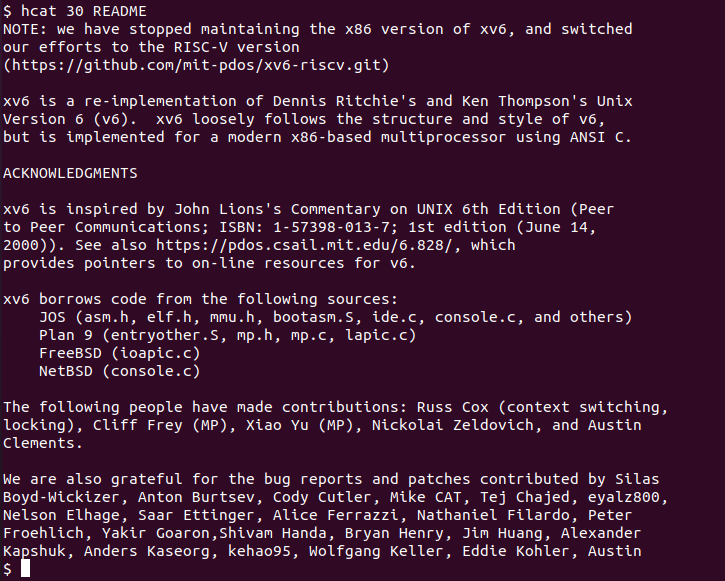


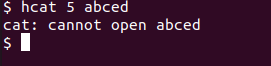
2. helloworld



3. hcat 다양한 줄 수 및 없는 파일

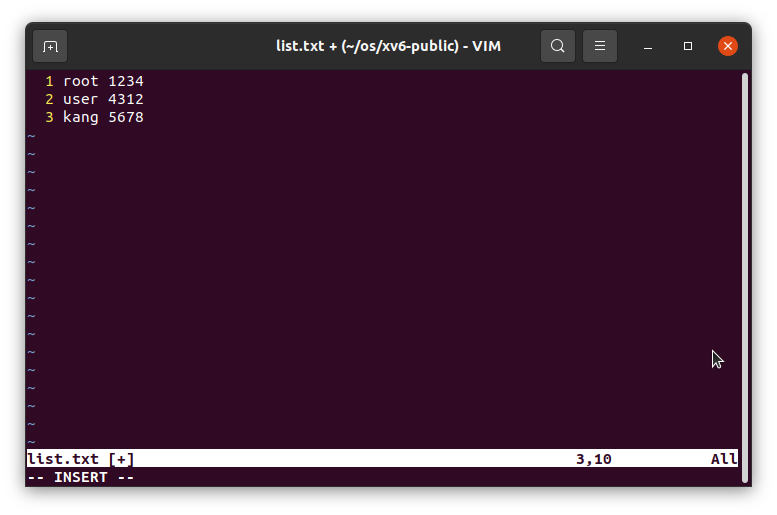


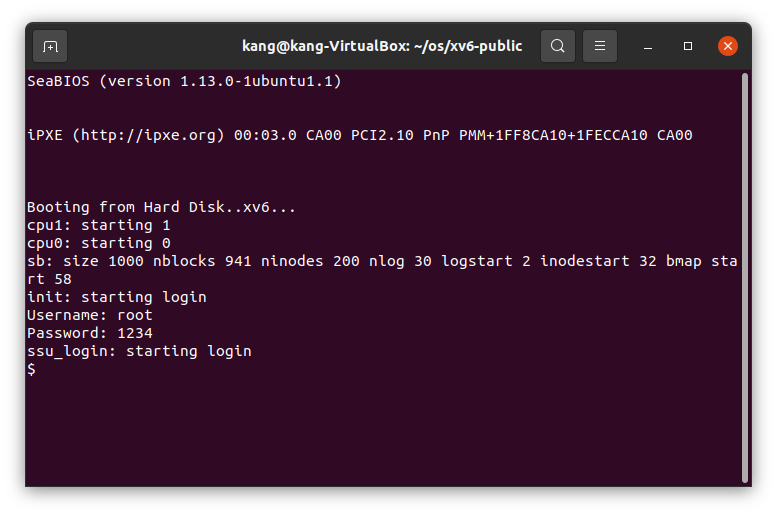




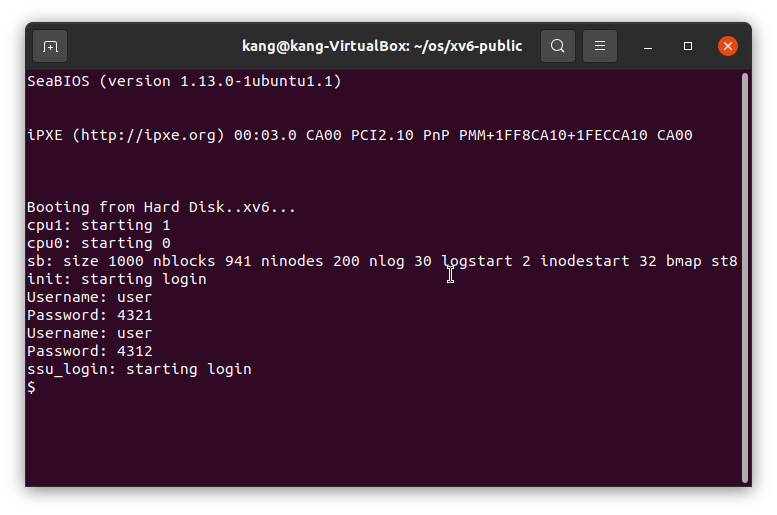
4.

list.txt



ssu\_login.c 로그인 성공

ssu\_login.c 로그인 실패 - >성공



**3.소스코드**

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\

    sleeplock.o\

    spinlock.o\

    string.o\

    swtch.o\

    syscall.o\

    sysfile.o\

    sysproc.o\

    trapasm.o\

    trap.o\

    uart.o\

    vectors.o\

    vm.o\

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

# TOOLPREFIX = i386-jos-elf

# Using native tools (e.g., on X86 Linux)

#TOOLPREFIX =

# Try to infer the correct TOOLPREFIX if not set

ifndef TOOLPREFIX

TOOLPREFIX := $(shell if i386-jos-elf-objdump -i 2>&1 | grep '^elf32-i386$$' >/dev/null 2>&1; \

    then echo 'i386-jos-elf-'; \

    elif objdump -i 2>&1 | grep 'elf32-i386' >/dev/null 2>&1; \

    then echo ''; \

    else echo "\*\*\*" 1>&2; \

    echo "\*\*\* Error: Couldn't find an i386-\*-elf version of GCC/binutils." 1>&2; \

    echo "\*\*\* Is the directory with i386-jos-elf-gcc in your PATH?" 1>&2; \

    echo "\*\*\* If your i386-\*-elf toolchain is installed with a command" 1>&2; \

    echo "\*\*\* prefix other than 'i386-jos-elf-', set your TOOLPREFIX" 1>&2; \

    echo "\*\*\* environment variable to that prefix and run 'make' again." 1>&2; \

    echo "\*\*\* To turn off this error, run 'gmake TOOLPREFIX= ...'." 1>&2; \

    echo "\*\*\*" 1>&2; exit 1; fi)

endif

# If the makefile can't find QEMU, specify its path here

# QEMU = qemu-system-i386

# Try to infer the correct QEMU

ifndef QEMU

QEMU = $(shell if which qemu > /dev/null; \

    then echo qemu; exit; \

    elif which qemu-system-i386 > /dev/null; \

    then echo qemu-system-i386; exit; \

    elif which qemu-system-x86\_64 > /dev/null; \

    then echo qemu-system-x86\_64; exit; \

    else \

    qemu=/Applications/Q.app/Contents/MacOS/i386-softmmu.app/Contents/MacOS/i386-softmmu; \

    if test -x $$qemu; then echo $$qemu; exit; fi; fi; \

    echo "\*\*\*" 1>&2; \

    echo "\*\*\* Error: Couldn't find a working QEMU executable." 1>&2; \

    echo "\*\*\* Is the directory containing the qemu binary in your PATH" 1>&2; \

    echo "\*\*\* or have you tried setting the QEMU variable in Makefile?" 1>&2; \

    echo "\*\*\*" 1>&2; exit 1)

endif

CC = $(TOOLPREFIX)gcc

AS = $(TOOLPREFIX)gas

LD = $(TOOLPREFIX)ld

OBJCOPY = $(TOOLPREFIX)objcopy

OBJDUMP = $(TOOLPREFIX)objdump

CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer

CFLAGS += $(shell $(CC) -fno-stack-protector -E -x c /dev/null >/dev/null 2>&1 && echo -fno-stack-protector)

ASFLAGS = -m32 -gdwarf-2 -Wa,-divide

# FreeBSD ld wants ``elf\_i386\_fbsd''

LDFLAGS += -m $(shell $(LD) -V | grep elf\_i386 2>/dev/null | head -n 1)

# Disable PIE when possible (for Ubuntu 16.10 toolchain)

ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]no-pie'),)

CFLAGS += -fno-pie -no-pie

endif

ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]nopie'),)

CFLAGS += -fno-pie -nopie

endif

xv6.img: bootblock kernel

    dd if=/dev/zero of=xv6.img count=10000

    dd if=bootblock of=xv6.img conv=notrunc

    dd if=kernel of=xv6.img seek=1 conv=notrunc

xv6memfs.img: bootblock kernelmemfs

    dd if=/dev/zero of=xv6memfs.img count=10000

    dd if=bootblock of=xv6memfs.img conv=notrunc

    dd if=kernelmemfs of=xv6memfs.img seek=1 conv=notrunc

bootblock: bootasm.S bootmain.c

    $(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c bootmain.c

    $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c bootasm.S

    $(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o

    $(OBJDUMP) -S bootblock.o > bootblock.asm

    $(OBJCOPY) -S -O binary -j .text bootblock.o bootblock

    ./sign.pl bootblock

entryother: entryother.S

    $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c entryother.S

    $(LD) $(LDFLAGS) -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o

    $(OBJCOPY) -S -O binary -j .text bootblockother.o entryother

    $(OBJDUMP) -S bootblockother.o > entryother.asm

initcode: initcode.S

    $(CC) $(CFLAGS) -nostdinc -I. -c initcode.S

    $(LD) $(LDFLAGS) -N -e start -Ttext 0 -o initcode.out initcode.o

    $(OBJCOPY) -S -O binary initcode.out initcode

    $(OBJDUMP) -S initcode.o > initcode.asm

kernel: $(OBJS) entry.o entryother initcode kernel.ld

    $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother

    $(OBJDUMP) -S kernel > kernel.asm

    $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > kernel.sym

# kernelmemfs is a copy of kernel that maintains the

# disk image in memory instead of writing to a disk.

# This is not so useful for testing persistent storage or

# exploring disk buffering implementations, but it is

# great for testing the kernel on real hardware without

# needing a scratch disk.

MEMFSOBJS = $(filter-out ide.o,$(OBJS)) memide.o

kernelmemfs: $(MEMFSOBJS) entry.o entryother initcode kernel.ld fs.img

    $(LD) $(LDFLAGS) -T kernel.ld -o kernelmemfs entry.o  $(MEMFSOBJS) -b binary initcode entryother fs.img

    $(OBJDUMP) -S kernelmemfs > kernelmemfs.asm

    $(OBJDUMP) -t kernelmemfs | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > kernelmemfs.sym

tags: $(OBJS) entryother.S \_init

    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 0 -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\

    \_forktest\

    \_grep\

    \_init\

    \_kill\

    \_ln\

    \_ls\

    \_mkdir\

    \_rm\

    \_sh\

    \_stressfs\

    \_usertests\

    \_wc\

    \_zombie\

    \_helloworld\

    \_hcat\

    \_ssu\_login\

fs.img: mkfs README list.txt $(UPROGS)

    ./mkfs fs.img README list.txt $(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 \

    xv6memfs.img mkfs .gdbinit \

    $(UPROGS)

# make a printout

FILES = $(shell grep -v '^\#' runoff.list)

PRINT = runoff.list runoff.spec README list.txt 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

    bochs -q

# try to generate a unique GDB port

GDBPORT = $(shell expr `id -u` % 5000 + 25000)

# QEMU's gdb stub command line changed in 0.11

QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \

    then echo "-gdb tcp::$(GDBPORT)"; \

    else echo "-s -p $(GDBPORT)"; fi)

ifndef CPUS

CPUS := 2

endif

QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)

qemu: fs.img xv6.img

    $(QEMU) -serial mon:stdio $(QEMUOPTS)

qemu-memfs: xv6memfs.img

    $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256

qemu-nox: fs.img xv6.img

    $(QEMU) -nographic $(QEMUOPTS)

.gdbinit: .gdbinit.tmpl

    sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@

qemu-gdb: fs.img xv6.img .gdbinit

    @echo "\*\*\* Now run 'gdb'." 1>&2

    $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)

qemu-nox-gdb: fs.img xv6.img .gdbinit

    @echo "\*\*\* Now run 'gdb'." 1>&2

    $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)

# CUT HERE

# prepare dist for students

# after running make dist, probably want to

# rename it to rev0 or rev1 or so on and then

# check in that version.

EXTRA=\

    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\

    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\

    printf.c umalloc.c helloworld.c hcat.c ssu\_login.c\

    README list.txt dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\

    .gdbinit.tmpl gdbutil\

dist:

    rm -rf dist

    mkdir dist

    for i in $(FILES); \

    do \

        grep -v PAGEBREAK $$i >dist/$$i; \

    done

    sed '/CUT HERE/,$$d' Makefile >dist/Makefile

    echo >dist/runoff.spec

    cp $(EXTRA) dist

dist-test:

    rm -rf dist

    make dist

    rm -rf dist-test

    mkdir dist-test

    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-rev10.tar.gz  # the next one will be 10 (9/17)

.PHONY: dist-test dist

helloworld.c 수정 없음

hcat.c

#include "types.h"

#include "stat.h"

#include "user.h"

char buf[512];

int line;

void

cat(int fd)

{

    int n;

    int i, count=0;

    char c;

    while((n = read(fd, buf, sizeof(buf))) > 0) {

        for(i=0; i<n; i++){

            if((c=buf[i])=='\n')

                count++;

            if(count==line)

                break;

        }

        if (write(1, buf, i+1) != i+1) {

            printf(1, "cat: write error\n");

            exit();

        }

        if(count==line)

            break;

    }

    if(n < 0){

        printf(1, "cat: read error\n");

        exit();

    }

}

int

main(int argc, char \*argv[])

{

    int fd;

    if(argc <= 2){

        cat(0);

        exit();

    }

    line=atoi(argv[1]);

    if((fd=open(argv[2], 0))<0){

        printf(1,"cat: cannot open %s\n", argv[2]);

        exit();

    }

    cat(fd);

    close(fd);

    exit();

}

init.c

// init: The initial user-level program

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

char \*argv[] = { "sh", 0 };

int

main(void)

{

  int pid, wpid;

  if(open("console", O\_RDWR) < 0){

    mknod("console", 1, 1);

    open("console", O\_RDWR);

  }

  dup(0);  // stdout

  dup(0);  // stderr

  for(;;){

    printf(1, "init: starting login\n");

    pid = fork();

    if(pid < 0){

      printf(1, "init: fork failed\n");

      exit();

    }

    if(pid == 0){

    exec("ssu\_login", argv);

    printf(1,"init: exec login failed\n");

    //exec("sh", argv);

//      printf(1, "init: exec sh failed\n");

    exit();

    }

    while((wpid=wait()) >= 0 && wpid != pid)

      printf(1, "zombie!\n");

  }

}

ssu\_login.c

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

char userID[16][32];

char pwdID[16][32];

char\* \_gets(char \*buf, int max)

{

  int i, cc;

  char c;

  for(i=0; i+1 < max; ){

    cc = read(0, &c, 1);

    if(cc < 1)

      break;

    buf[i++] = c;

    if(c == '\n' || c == '\r')

      break;

  }

  i--;

  buf[i] = '\0';

  return buf;

}

int tokenizing(char\* buf, int t, char\* target){

    int i;

    int j=0;

    for(i=t; i<1024;i++ ){

        target[j++]=buf[i];

        if(buf[i]==' '||buf[i]=='\n'||buf[i]=='\0')

            break;

    }

    j--;

    //printf(1,"===================t %d", i);

    target[j]='\0';

    return ++i;

}

void get\_user\_list(){

    int fd, i, t=0;

    int c;

    char buf[1024];

    fd=open("list.txt", O\_RDONLY);

    if((c=read(fd, buf, sizeof(buf)))<0)

    {

        printf(1,"read list.txt error\n");

        exit();

    }

    if(c==0)

        return;

    for(i=0;i<10;i++){

        // list.txt 에서 Username, Password 정보를 userID, pwdID에 저장

        //memset(userID[i],0, 32);

        //memset(pwdID[i],0, 32);

        t=tokenizing(buf,t, userID[i]);

        t=tokenizing(buf,t, pwdID[i]);

    //  printf(1,"i:   user %s    pwd %s\n", userID[i], pwdID[i]);

    }

}

int check\_idpw(){

    //입력받은 Username, Password와 list.txt 비교

    int i;

    char username[32];

    char password[32];

    printf(1,"Username: ");

    \_gets(username, sizeof(username));

    printf(1,"Password: ");

    \_gets(password, sizeof(password));

//  printf(1,"check u: %s, p: %s \n", username, password);

    for(i=0;i<10;i++){

        if(!strcmp(username, userID[i]))

            if(!strcmp(password, pwdID[i])){

                return 1;

            }

    }

    return 0;

}

int main(int argc, char \*argv[])

{

    int pid, wpid;

    get\_user\_list();

    while(1){

        if(check\_idpw()){

            for(;;){

                printf(1, "ssu\_login: starting login\n");

                pid = fork();

                if(pid < 0){

                    printf(1, "ssu\_login: fork failed\n");

                    exit();

                }

                if(pid == 0){

                    exec("sh", argv);

                    printf(1,"ssu\_login: exec sh failed\n");

                    exit();

                }

                while((wpid=wait()) >= 0 && wpid != pid)

                    printf(1, "zombie!\n");

            }

            exit();

        }

    }

    exit();

}