컴퓨터학부 20201853 서민비

1. 개요

1) xv6와 QEMU를 다운로드 및 설치 후 컴파일/실행한 화면을 캡쳐한다.

2-가)Hello World xv6를 출력하는 helloworld.c 프로그램을 작성하고, Makefile에 쉘 프로그램을 추가하여 make시 컴파일 되도록 변경한다. 그리고 xv6 실행 후 helloworld 명령어 실행 시 화면을 캡쳐한다.

2-나)파일의 n번째 행을 터미널에 출력하는 hcat.c 프로그램을 작성하고, Makefile에 쉘 프로그램을 추가하여 make시 컴파일 되도록 변경한다. 그리고 xv6 실행 후 hcat 명령어 실행 시 화면을 캡쳐한다.

3)부팅 시 username과 password를 입력 받아 list.txt 파일에 입력 받은 username과 password가 존재하는지 확인 후 존재한다면, shell 프로그램을 호출하는 로그인 프로세스를 구현한다.

2. 결과(실행결과 및 설명)

-1)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 모니터, 검은색이(가) 표시된 사진

자동 생성된 설명

-2 –가)

텍스트이(가) 표시된 사진

자동 생성된 설명

-2 – 나)

텍스트이(가) 표시된 사진

자동 생성된 설명

hcat 5 README 입력 시 출력 화면 캡쳐

텍스트이(가) 표시된 사진

자동 생성된 설명

hcat 1 README 입력 시 출력 화면 캡쳐

텍스트이(가) 표시된 사진

자동 생성된 설명

hcat 51 README 입력 시 출력 화면 캡쳐

-3 )

텍스트이(가) 표시된 사진

자동 생성된 설명

부팅 시 화면

텍스트이(가) 표시된 사진

자동 생성된 설명

list.txt 파일에 저장된 username, password

텍스트이(가) 표시된 사진

자동 생성된 설명

올바른 username과 password를 입력으로 받았을 시 shell 프로그램 실행

텍스트이(가) 표시된 사진

자동 생성된 설명

List.txt파일에 존재하지 않는 username, password 입력 시 username, password 재입력

텍스트이(가) 표시된 사진

자동 생성된 설명

List.txt파일에 존재하지 않는 username, password 입력 후 올바른 username, password 재입력

텍스트이(가) 표시된 사진

자동 생성된 설명

List.txt파일에 존재하지 않는 username, password 입력 후 올바른 username, password 재입력

3. 소스코드

Helloworld.c

#include "types.h"

#include "stat.h"

#include "user.h"

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

{

    printf(1,"Hello World XV6\n");

    exit();

}

Hcat.c

#include "types.h"

#include "stat.h"

#include "user.h"

char buf;

int row;

void

cat(int fd)

{

  int n;

  int cnt=1;

  while((n = read(fd, &buf, sizeof(buf))) > 0) {  //read one char in file

    if((cnt==row)&&(write(1, &buf, n) != n)) { //then print

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

        exit();

      }

    if(buf!='\n'){  //if char is not enter

      ;

    }else{ //if char is enter

      cnt++;

    }

  }

  if(n < 0){

    printf(1, "cat: read error\n");

    exit();

  }

}

int

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

{

  int fd;

  if(argc <= 1){

    cat(0);

    exit();

  }

  if(argc!=3){

    printf(1, "input error\n");

    exit();

  }

    if((fd = open(argv[2], 0)) < 0){

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

      exit();

    }

    char \*s=argv[1];

    while('0' <= \*s && \*s <= '9')

      row = row\*10 + \*s++ - '0';

    printf(1, "argv[0]: %s,argv[1]: %s,argv[2]: %s, row : %d\n", argv[0], argv[1], argv[2],strlen(argv[1]));

    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

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

    pid = fork(); //스레드생성. 성공 시 자식 프로세스 생성 후 부모 프로세서와 함께 돌아감.

    if(pid < 0){

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

      exit();

    }

    if(pid == 0){ //자식 프로세스에서 돌아가는 코드

      exec("ssu\_login",argv);

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

      exit();

    }

    while((wpid=wait()) >= 0 && wpid != pid)  //자식프로세스가 종료될 떄까지 부모가 while문 돌면서 기다림

      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 \*username;

char \*password;

void get\_user\_list(){

    int fd;

    if((fd=open("list.txt",O\_RDONLY))<0){

        printf(1, "login.txt can't open\n");

        exit();

    }

    for(int j=0;j<10;j++){

        int i, cc;

        char c;

        char buf[100];

        for(i=0; i+1 < 32; ){

            cc = read(fd, &c, sizeof(c));

            if(cc < 1)

                break;

            if(c==' ')

                break;

            buf[i++] = c;

        }

        buf[i] = '\0';

        strcpy(userID[j],buf);

        for(i=0; i+1 < 32; ){

            cc = read(fd, &c, sizeof(c));

            if(cc < 1)

                break;

            if(c == '\n' || c == '\r')

                break;

            buf[i++] = c;

        }

        buf[i] = '\0';

        strcpy(pwdID[j],buf);

    }

    /\*int i,j, cc;

    char c;

    while(1){

        for(i=0; i+1 < 1024; ){

            cc = read(fd, &c, sizeof(c));    //read(fd, &buf, sizeof(buf))

            if(cc < 1)  //end file

                break;

            if(c==' ')

                break;

            userID[i++] = c;

        }

        if(cc < 1)  //end file

            break;

        for(j=0;j+1<1024; ){

            cc = read(fd, &c, sizeof(c));    //read(fd, &buf, sizeof(buf))

            if(cc < 1)  //end file

                break;

            if(c == '\n' || c == '\r')  //end line

                break;

            pwdID[j++] = c;

        }

        if(cc < 1)  //end file

            break;

        if(c == '\n' || c == '\r'){

            userID[i] = '\0';

            pwdID[j]='\0';

        }

    }\*/

}

int check\_idpw(){

    //입력받은 username, password와 list.txt 비교

    //로그인 성공이면 1리턴, 실패면 0리턴

    for(int j=0;j<10;j++){

        if((strcmp(userID[j],username)==0)&&(strcmp(pwdID[j],password)==0)){

            return 1;

        }

    }

    return 0;

}

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

    get\_user\_list();

    /\*for(int z=0;z<16;z++){

        printf(1,"%s %s\n",userID[z], pwdID[z]);

    }\*/

    int login=1;

    while(login){

        printf(1,"Username: ");

        username = gets("username", 32);

        username[strlen(username)-1]='\0';  //엔터 버리기

        printf(1,"Password: ");

        password = gets("password", 32);

        password[strlen(password)-1]='\0';  //엔터 버리기

        //printf(1,"%s %s\n",username, password);

        if(check\_idpw() ==1){

            //exec("sh", argv);

            //printf(1, "login success\n");

            login=0;

        }else{

            //printf(1, "login fail\n");

        }

    }

    exec("sh", argv);

    return 0;

}

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