20192392 컴퓨터학부 김한승 HS9903(중간 / 기말 비밀번호)

1. 개요

과제 세부 목표로 3가지를 수행합니다, 첫번째로, 과제 목표로 제시된 xv6 설치 및 컴파일을 수행합니다.

xv6를 git clone 명령어를 통해 다운로드 한 후, xv6에는 vi / vim 또는 gcc 컴파일러가 없으므로, 리눅스 환경에서 xv6 프로그램을 작성하고 make qemu를 통해 컴파일 후, 생성된 실행 파일을 xv6 상에서 수행시킵니다.

helloxv6 / htac 명령어 수행을 위해 Makefile을 따로 수정합니다.(UPROGS / EXTRA)

두번째로, “Hello xv6 World”를 출력하는 “helloxv6” 명령어를 위하여 helloxv6.c를 구현합니다.

세번째로, <Usage: htac line file\_name> file\_name에 해당하는 파일을 열어 해당 파일에 맨 뒤 줄부터 line 수만큼의 줄을 역순으로 출력하는 “htac” 명령어를 위하여 htac.c를 구현합니다.

3번 항목 실행 결과를 위한 실행 테스트에 대한 개요입니다.

첫번째, 설치 및 컴파일을 마친 후 파일 / 디렉토리의 리스트를 출력하기 위해 내부 명령어 ls를 실행시켜 테스트합니다.

두번째, helloxv6 내부 명령어 수행을 위해 Makefile에 UPROGS / EXTRA를 수정한 후, helloxv6.c 파일에 printf() 문을 통해 표준 출력 창에 “Hello xv6 World”를 출력하여 테스트합니다.

세번째, cat 명령어를 구현한 cat.c를 참고하여 htac.c를 구현합니다. 해당 명령어를 통해 역순으로 해당 파일의 해당하는 line 수만큼의 줄을 역순으로 출력합니다. 테스트 케이스에서는 README 파일을 0줄 / 1줄 / 5줄 / 51줄(파일의 전체 길이) 만큼 출력하는 테스트를 하였습니다.

2. 상세 설계

2-1. helloxv6.c

2-1-1. int main(int argc, char \*argv[])

printf(1, “Hello xv6 World\n”); 를 통해 표준 출력 창에 해당 문장을 출력합니다.

텍스트, 스크린샷, 원, 폰트이(가) 표시된 사진

자동 생성된 설명

2-2. htac.c

2-2-1. int main(int argc, char \*argv[])

argc < 3 인 경우 Usage를 출력하도록 하였습니다. (Usage : htac line file\_name)

argv[1]에 해당하는 char 배열을 int형으로 변환하였습니다.

argv[2]에 해당하는 file\_name을 open하여 fd(파일 디스크립터)를 얻고, 전역 변수 fileName에 해당 인자를 복사하였습니다. 그 후, htac(fd)를 호출하고, 호출이 끝나면 파일 디스크립터를 닫고 프로그램을 종료합니다.

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

2-2-2. void htac(int fd);

첫번째로, 전체 파일을 read하여서 fileLine이 몇 줄인지 구합니다.

tempFd를 통해 main() 함수에서 strcpy()를 통해 복사하였던 파일을 open하여서 buf2[1]를 통해

출력하지 않을 문자들은 모두 읽습니다.(51줄의 파일 중 5줄을 출력한다고 했을 경우 46번째 줄까지 모두 읽기)

해당 read작업을 통해 tempFd 파일 디스크립터의 파일 오프셋은 역순 출력을 할 때 마지막으로 출력할 라인의 첫번째 문자를 가리키게 됩니다.

만약, line(프로그램 인자로 받은 역순으로 출력할 라인 수) 와 fileLine(역순으로 출력할 파일의 전체 라인 수)가 같을 경우에는 파일 오프셋을 옮기는 작업을 하지 않습니다. why? 전체 출력을 진행해야 하기 때문입니다.

read()를 통해 tempFd의 역순출력을 위해 출력할 문장들을 버퍼에 읽습니다.

버퍼에서 start 와 end 인덱스를 옮겨 가며 마지막 줄에서부터 역순으로 출력할 마지막 줄(마지막 줄로부터 line 수 만큼 위에 줄) 을 출력합니다.

현재 줄의 출력은 start+1~end-1 출력을 반복적으로 수행합니다. 이 때, start +1는 현재 출력할 문장의 첫번째 문자를 가리키며, end-1은 출력할 문장의 마지막을 가리킵니다.

하나의 문자열을 출력하였다면, 개행 문자(‘\n’)를 출력하고 start를 감소시킨 후 다시 출력할 문장이 있는지 체크합니다.

모든 역순 출력이 완료되면(파일의 마지막 줄 ~ 파일의 마지막 줄 – line +1 줄) 였던 tempFd를 닫고 함수를 종료합니다.

텍스트, 스크린샷, 도표, 원이(가) 표시된 사진

자동 생성된 설명

스크린샷, 도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

3. 실행 결과

3-1. make qemu 실행 결과 및 ls 실행 결과

3-1-1. make qemu

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

3-1-2. ls

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

3-2. helloxv6 실행 결과

3-2-1. helloxv6

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

3-3. htac 실행 결과

3-3-1. htac 0 README

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

3-3-2. htac 1 README

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

3-3-2. htac 5 README

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3-3-4. htac 51 README

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(이어지는 출력임을 표시하기 위해 Eldar ~ 라인을 겹치게 캡쳐 했습니다. 51개 줄이므로 화면에 다 표시되지 못하기 때문입니다.)

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

3-3-5. htac README

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

3-3-6. htac 5

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

4. 소스 코드

4-1. helloxv6.c

#include "types.h"

#include "stat.h"

#include "user.h"

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

{

    printf(1, "Hello xv6 World\n"); // 표준 출력에 Hello xv6 World 출력

    exit();

}

4-2. htac.c

#include "types.h"

#include "stat.h"

#include "user.h"

#define BUFFER\_SIZE 2048 + 256 // 버퍼 크기

#define NAME\_SIZE   256 // 파일 네임 크기

int line; // 역순으로 출력할 라인 수 전역 변수

char buf[BUFFER\_SIZE]; // 버퍼

char fileName[NAME\_SIZE]; // void htac(int fd)에서 tempFd 파일 디스크립터에 오픈하기 위해 전역 변수 저장

void htac(int fd)

{

    int n,i;

    int tempFd;

    int fileLine = 0;

    // 파일 라인수를 체크

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

    {

        for(i = 0; i < n; i++)

        {

            if(buf[i] == '\n') fileLine++;

        }

    }

    if(n < 0)

    {

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

        exit();

    }

    // tempFd에 역순으로 출력할 파일을 오픈

    if((tempFd = open(fileName, 0)) < 0)

    {

        printf(1, "htac : canoot 2nd open %s\n",fileName);

        exit();

    }

    int index = 0; // 현재 파일 오프셋

    char buf2[1]; // 파일 오프셋을 옮기기 위해 1문자씩 읽기 위한 버퍼2

    // 출력하지 않을 줄은 read를 통해 파일 오프셋을 옮긴다.

    // line 과  fileLine 같을 때는 파일 오프셋을 옮기지 않는다.(전체 출력)

    if(line != fileLine)

    {

        while((n = read(tempFd, buf2, 1)) > 0)

        {

            if(buf2[0] == '\n') index++;

            if(index == (fileLine - line)) break;

        }

    }

    // 버퍼 ""으로 값 복사 및 역순으로 출력할 문자열들을 read 한다.

    strcpy(buf, "");

    n = read(tempFd, buf,sizeof(buf));

    if(n < 0)

    {

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

        exit();

    }

    // start가 '\n'가 될때마다 혹은 start가 음수가 되면  해당 라인을 출력한다.

    // start+1 ~ end-1 까지 출력 후 개행 문자 출력

    int start = n-2,end = n-1;

    while(start >= 0)

    {

        // 출력할 문자열의 처음을 가리키기 위해 start--

        // 파일에 맨 처음으로 가면 -1(start)  start+1 == 0(파일의 처음)

        // 파일 중간에 '\n'을 만난다. 출력할 문자열의 처음은 start+1

        while(start >= 0 && buf[start] != '\n')

            start--;

        for(i = start+1; i < end; i++)

        {

            buf2[0] = buf[i];

            if(write(1,buf2,1) != 1)

            {

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

                exit();

            }

        }

        printf(1, "\n"); // 개행 문자 출력

        // 출력을 완료하고, end를 start로 옮겨줍니다.

        end = start;

        // 다음 문자열 출력을 위해 start인덱스를 -1 합니다.

        start--;

    }

    close(tempFd);

}

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

{

    int fd,i;

    if(argc < 3)

    {

        // Usage와 동일한 사용을 하지 않은 경우 인자가 3개 미만이라면 Usage 출력

        printf(1, "Usage : %s line file\_name\n",argv[0]);

        exit();

    }

    // 역순으로 출력하기 위한 line수 인자를 int형으로 계산한다. atoi 사용대신 구현

    line = 0;

    for(i = 0; i < strlen(argv[1]); i++)

    {

        line \*= 10;

        line += argv[1][i] - '0';

    }

    // htac에 인자로 전달할 fd를 open()을 통해 저장한다.

    if((fd = open(argv[2], 0)) < 0)

    {

        printf(1,"htac : catnnot 1st open %s\n",argv[2]); // 파일 오픈 에러 처리

        exit();

    }

    strcpy(fileName, argv[2]);

    htac(fd);

    close(fd);

    exit();

}

4-3. 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\

    \_helloxv6\

    \_htac\

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 \

    xv6memfs.img 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

    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 helloxv6.c htac.c\

    README 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