

1. 개요

본 설계 과제는 xv6라는 교육용 운영체제를 설치(설계과제 1-1), 컴파일 및 수정하여 특정 응용 프로그램(설계과제 1-2, 1-3)을 구현하는 것을 목표로 하고 있다.

구체적으로, xv6는 MIT에서 개발한 교육용 운영체제로, UNIX V6와 ANSI C를 기반으로 한다. 기본적으로 리눅스나 BSD와 같은 현대의 운영체제보다는 기능이 단순하나, 운영체제의 주요 개념과 구조를 학습하기에 적합하다.

또한, xv6 상에는 텍스트 편집기나 컴파일러가 없기 때문에, 작업을 수행하기 위해서 개발자의 리눅스 환경에서 프로그램을 작성하고 컴파일한 후, 그 결과로 생성된 실행 파일을 xv6에서 실행해야 한다. 이렇게 다른 곳에서 프로그램을 컴파일하는 것을 'Cross Compile'이라고 한다.

결론적으로, 본 설계 과제는 xv6 운영체제의 기본적인 설치 및 컴파일 방법을 학습하고, 간단한 응용 프로그램을 구현하여 운영체제의 동작과 프로그래밍을 이해하는 것을 목표로 한다.

설계과제 1-2) helloxv6 응용 프로그램:

이 프로그램은 "Hello xv6 World"라는 메시지를 출력하는 프로그램이다. "helloxv6.c"라는 파일로 작성된다.

설계과제 1-3) htac 응용 프로그램:

이 프로그램은 사용자로부터 htac에 더하여 두 가지 인자를 받아들인다. 첫 번째 인자는 출력할 라인 수, 두 번째 인자는 해당 라인 수를 파일의 제일 마지막 줄부터 역순으로 출력할 파일이 된다. 따라서, 사용자가 5라는 숫자와 "README"라는 파일을 입력하면, "README"의 마지막 5줄을 역순으로 출력하게 된다. 이 프로그램은 "htac.c"라는 파일로 작성된다.

두 프로그램 구현을 위한 준비 작업으로 우선 **설계과제 1-1인 “xv6 설치 및 컴파일”** 단계가 요구된다.

xv6 설치 및 컴파일:

먼저, xv6의 소스 코드를 다운로드해야 한다. QEMU x86이라는 에뮬레이터를 설치해야 한다. xv6는 이 에뮬레이터 위에서 실행되기 때문인데, xv6는 실제 하드웨어에서도 동작하지만 수정 및 테스트를 위해 QEMU x86 에뮬레이터 사용이 권장된다.

QEMU를 설치하는 다음 명령어를 통하여 진행된다:

```
$ apt-get install qemu-kvm
```

이후, xv6 소스 코드를 컴파일하고 QEMU 에뮬레이터에서 실행한다.

```
$ make
```

```
$ make qemu
```

설계 과제 1-2는 xv6 운영체제 상에서 작동하는 응용 프로그램인 "helloxv6"을 작성하는 것에 중점을 둔다.

helloxv6 응용 프로그램 작성:

목표는 "Hello xv6 World"라는 메시지를 출력하는 프로그램인 "helloxv6.c"를 작성하는 것이다. 제시된 예시 코드와 같이 xv6에서 필요한 헤더 파일들을 포함하여, main 함수 내에서 printf를 사용하여 메시지를 출력하고, exit 함수로 프로그램을 종료한다.

Makefile 수정:

프로그램을 컴파일하려면 "helloxv6.c" 파일이 Makefile에서 인식되어야 한다. "UPROGS"에는 사용자 프로그램의 실행 파일 이름이 나열되어 있고, 이곳에 "_helloxv6"를 추가해준다. "EXTRA"에는 추가적인 파일들이 나열되어 있으며, 이 곳에도 "helloxv6.c"를 추가한다.

xv6 컴파일 및 helloxv6 실행:

Makefile 수정 후, xv6를 컴파일하면 "helloxv6" 응용 프로그램도 함께 컴파일된다. 컴파일이 완료되면, xv6를 시작하고 "helloxv6" 명령어를 실행하여 "Hello xv6 World" 메시지가 정상적으로 출력되는지 확인한다.

설계 과제 1-3는 "htac"이라는 셸 프로그램을 중점적으로 다룬다. 이 프로그램은 사용자가 지정한 줄 수만큼 파일의 마지막 행부터 역순으로 터미널에 출력하는 기능을 한다.

htac 셸 프로그램 작성:

1. "htac.c"라는 파일 내에서 해당 프로그램을 구현한다.
2. 기존의 "cat.c" 파일을 참고하여, "htac(int fd)"라는 함수를 구현한다.
3. 주어진 파일 디스크립터에서 파일의 내용을 역순으로 출력하는 역할을 한다.
4. "int line;"이라는 전역 변수가 "htac.c" 파일 내에서 선언되어야 한다. (본인은 이 변수를 최종적으로 저장된 줄 수(출력 가능한 최대 줄 수)로 사용하였다.)

htac 실행:

설계된 "htac.c"를 컴파일하여 xv6 내에서 실행 가능한 상태로 전환한 뒤, "htac" 명령어를 실행하여 제대로 동작하는지 확인한다.

제공된 "htac.c"의 "main" 함수 예시를 통해 알 수 있듯이, 함수는 파일 디스크립터를 열고, 주어진 행의 수만큼 파일의 내용을 마지막 줄부터 역순으로 출력한 후 프로그램을 종료하게 된다. 이 구조를 참고하여 "for" 루프와 파일 디스크립터 관련 코드, "htac" 함수 호출 등의 세부 내용을 추가한다.

2. 상세설계

helloxv6.c 파일은 예시와 동일한 코드 사용하였다.

htac.c는 xv6 시스템의 일부가 아닌 xv6 시스템 호출(파일 작업(open, read, close 등) 및 메모리 관리(malloc, free))을 활용한 프로그램으로써 설계과제 1-3의 조건들을 만족시키며 설계하였고, 호출과정은 아래와 같다.

htac에 대한 간략한 호출 과정 :

1. main()은 프로그램 시작
 2. 명령어 라인 인자 체크
 3. open 호출을 사용하여 파일 열기
 4. 파일 디스크립터를 통해 htac(fd) 호출
 5. read 호출로 파일 읽기
 6. malloc을 사용하여 파일의 줄을 메모리에 저장
 7. 목표된 라인을 printf 함수를 사용해 출력
 8. free를 사용하여 메모리 해제
 9. 프로그램 종료
- (더 자세한 설계 설명은 “4. 수정한 소스코드”의 주석 부분에 추가하였다.)

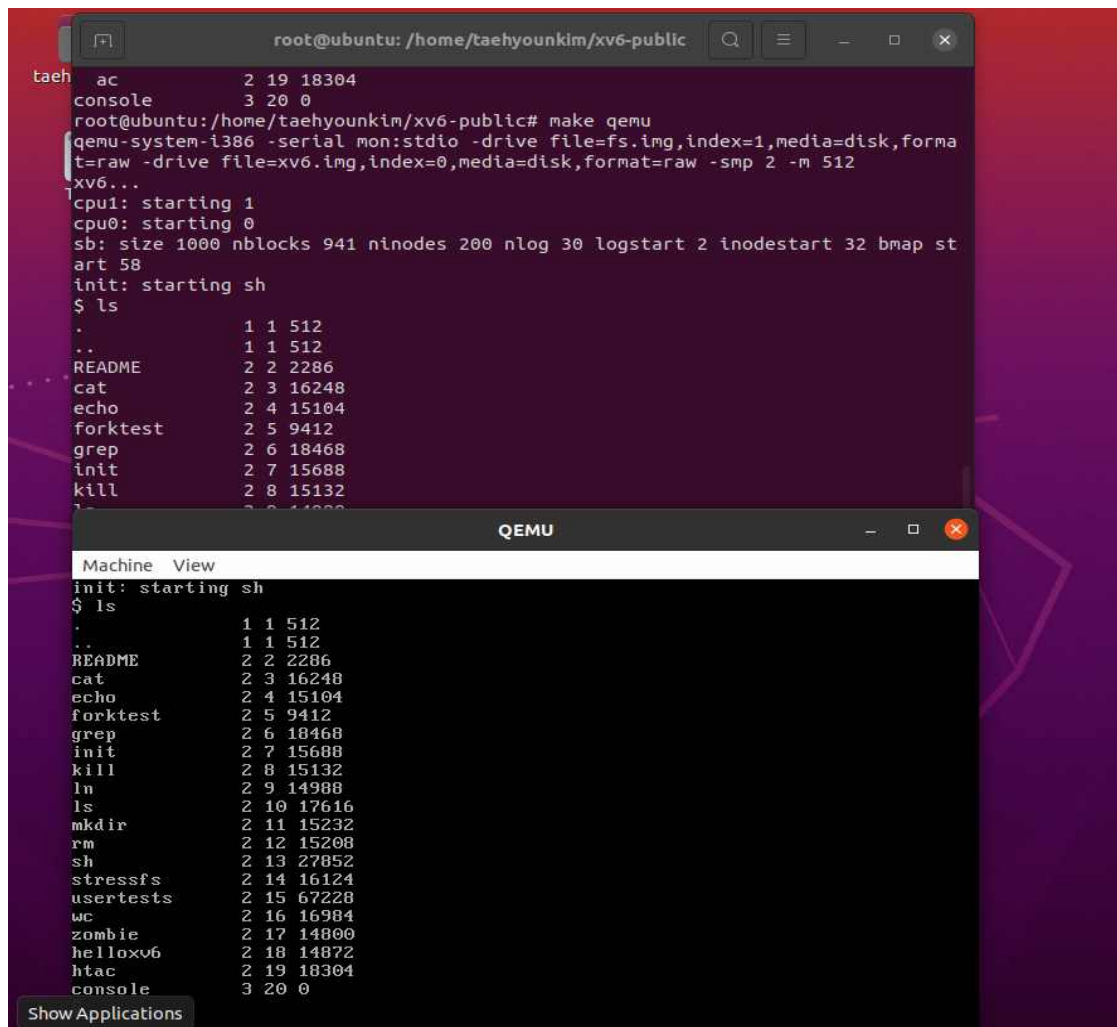
Makefile 수정 내용 : _helloxv6와 _htac를 UPROGS에 추가, helloxv6.c, htac.c를 EXTRA에 추가)

**수업 댓글에서도 질문하였지만 UPROGS에만 추가하였을 때도 정상작동하여 EXTRA에도 이를 추가하는 것이 어떠한 차이가 있는지 의문이다.

스스로 찾아본 바로는 'UPROGS', 'EXTRA' 두 곳 모두에 포함시켰을 경우, 메인 xv6 일부로 빌드되고, 또한 EXTRA 유틸리티의 일부로 빌드되어 명령어가 두 번 빌드되기 때문에 동일한 명령어를 두 곳 모두 포함시키는 것이 빌드 과정 중 비효율을 발생시킬 수 있다고 한다.

3. 결과(다양한 입력에 따른 실행 결과 등을 캡처 및 해석 등 포함)

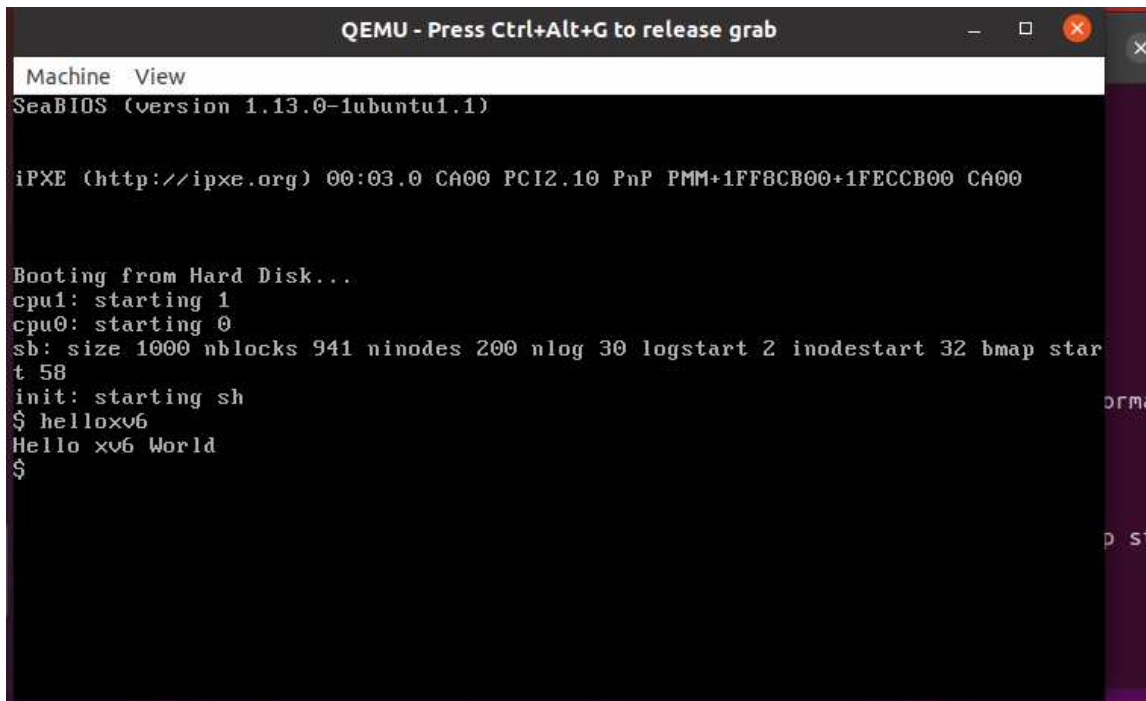
1) xv6 설치 후 ls 실행 결과



```
root@ubuntu: /home/taehyounkim/xv6-public
taeh ac          2 19 18304
console        3 20 0
root@ubuntu:/home/taehyounkim/xv6-public# make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat        2 3 16248
echo       2 4 15104
forktest   2 5 9412
grep       2 6 18468
init       2 7 15688
kill       2 8 15132
ln         2 9 14988
ls         2 10 17616
mkdir      2 11 15232
rm         2 12 15208
sh         2 13 27852
stressfs   2 14 16124
usertests  2 15 67228
wc         2 16 16984
zombie     2 17 14800
helloxv6   2 18 14872
htac       2 19 18304
console    3 20 0
```

예시와 동일하게 정상적으로 작동하였음.

2) helloxv6 실행결과



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

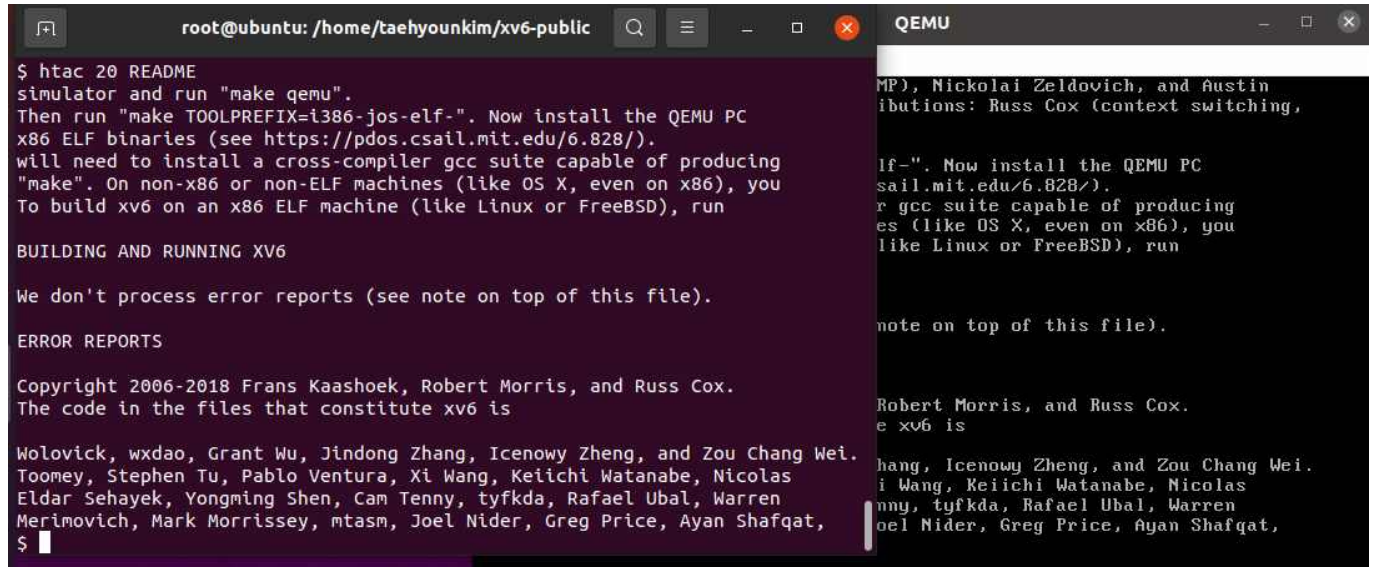
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ helloxv6
Hello xv6 World
$
```

예시와 동일하게 정상적으로 작동하였음.

3) htac 실행결과

“htac n <filename>” 에서 n는 ($0 < n \leq 1024$, n은 정수형)로 제한한다.

ex) htac 20 README 실행 결과 (파일의 제일 마지막줄부터 역순으로 20줄을 출력한다.)



```
root@ubuntu: /home/taehyounkim/xv6-public QEMU
$ htac 20 README
simulator and run "make qemu".
Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC
x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
will need to install a cross-compiler gcc suite capable of producing
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run

BUILDING AND RUNNING XV6

We don't process error reports (see note on top of this file).

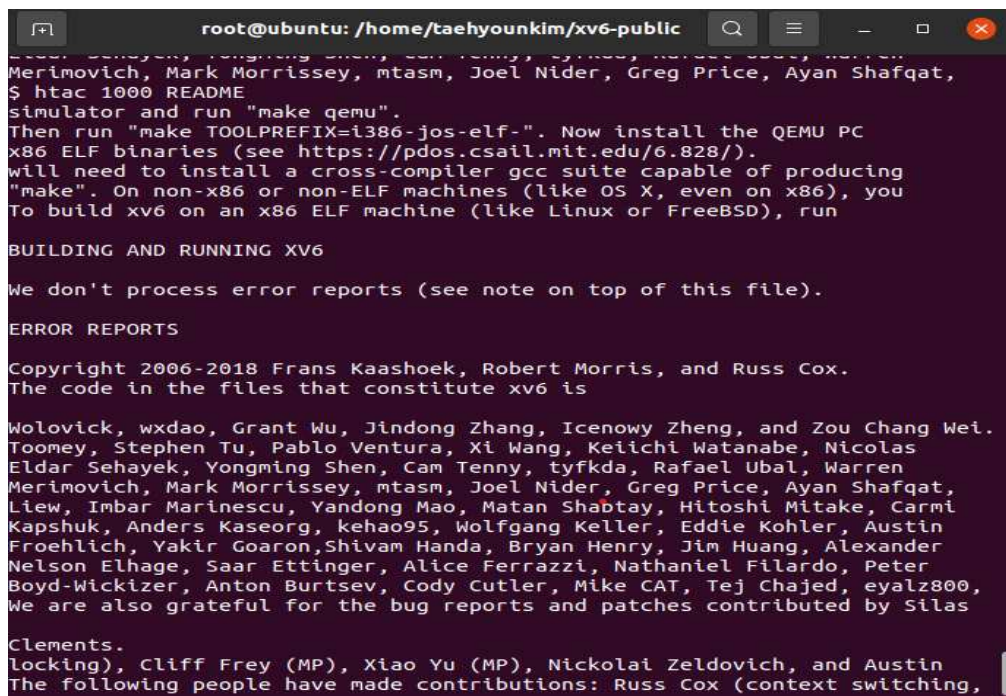
ERROR REPORTS

Copyright 2006-2018 Frans Kaashoek, Robert Morris, and Russ Cox.
The code in the files that constitute xv6 is

Wolovick, wxdao, Grant Wu, Jindong Zhang, Icenowy Zheng, and Zou Chang Wei.
Toomey, Stephen Tu, Pablo Ventura, Xi Wang, Keiichi Watanabe, Nicolas
Eldar Sehayek, Yongming Shen, Cam Tenny, tyfkda, Rafael Ubal, Warren
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
$
```

해당 파일의 줄 수를 초과해서 요청시에는 파일 내용 전체 줄을 역순으로 출력하고 정상 종료한다.

ex) htac 1000 README 실행결과 (상단부 캡처) =>(1000줄 이하의 파일이므로 전체를 역순으로 출력한다.)



```
root@ubuntu: /home/taehyounkim/xv6-public
...
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
$ htac 1000 README
simulator and run "make qemu".
Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC
x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
will need to install a cross-compiler gcc suite capable of producing
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run

BUILDING AND RUNNING XV6

We don't process error reports (see note on top of this file).

ERROR REPORTS

Copyright 2006-2018 Frans Kaashoek, Robert Morris, and Russ Cox.
The code in the files that constitute xv6 is

Wolovick, wxdao, Grant Wu, Jindong Zhang, Icenowy Zheng, and Zou Chang Wei.
Toomey, Stephen Tu, Pablo Ventura, Xi Wang, Keiichi Watanabe, Nicolas
Eldar Sehayek, Yongming Shen, Cam Tenny, tyfkda, Rafael Ubal, Warren
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
Liew, Imbar Marinescu, Yandong Mao, Matan Shabtay, Hitoshi Mitake, Carmi
Kapshuk, Anders Kaseorg, kehao95, Wolfgang Keller, Eddie Kohler, Austin
Froehlich, Yakir Goaron, Shivam Handa, Bryan Henry, Jim Huang, Alexander
Nelson Elhage, Saar Ettinger, Alice Ferrazzi, Nathaniel Filardo, Peter
Boyd-Wickizer, Anton Burtsev, Cody Cutler, Mike CAT, Tej Chajed, eyalz800,
We are also grateful for the bug reports and patches contributed by Silas
Clements.
locking), Cliff Frey (MP), Xiao Yu (MP), Nikolai Zeldovich, and Austin
The following people have made contributions: Russ Cox (context switching,
```


ex) htac 1000 README 실행결과 (하단부 캡처) =>(1000줄 이하의 파일이므로 전체를 역순으로 출력한다. 마지막으로 출력된 줄이 README 파일의 첫번째 줄임을 확인할 수 있다.)

```
root@ubuntu: /home/taehyounkim/xv6-public
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
Liew, Imbar Marinescu, Yandong Mao, Matan Shabtay, Hitoshi Mitake, Carmi
Kapshuk, Anders Kaseorg, kehao95, Wolfgang Keller, Eddie Kohler, Austin
Froehlich, Yakir Goaron, Shivan Handa, Bryan Henry, Jim Huang, Alexander
Nelson Elhage, Saar Ettinger, Alice Ferrazzi, Nathaniel Filardo, Peter
Boyd-Wickizer, Anton Burtsev, Cody Cutler, Mike CAT, Tej Chajed, eyalz800,
We are also grateful for the bug reports and patches contributed by Silas
Clements.
locking), Cliff Frey (MP), Xiao Yu (MP), Nickolai Zeldovich, and Austin
The following people have made contributions: Russ Cox (context switching,
NetBSD (console.c)
FreeBSD (ioapic.c)
Plan 9 (entryother.S, mp.h, mp.c, lapic.c)
JOS (asm.h, elf.h, mmu.h, bootasm.S, ide.c, console.c, and others)
xv6 borrows code from the following sources:
provides pointers to on-line resources for v6.
2000)). See also https://pdos.csail.mit.edu/6.828/, which
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14,
xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
ACKNOWLEDGMENTS
but is implemented for a modern x86-based multiprocessor using ANSI C.
Version 6 (v6). xv6 loosely follows the structure and style of v6,
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
(https://github.com/mit-pdos/xv6-riscv.git)
our efforts to the RISC-V version
NOTE: we have stopped maintaining the x86 version of xv6, and switched
$
```

“htac n <filename>” 에서 n이 정수가 아니라면 경고문 출력 후 종료한다.

ex) htac 10.3 README 실행결과 =>(10.3은 정수가 아니므로 1~1024인 정수를 입력할 것을 알린다.)

```
taehyounkim@ubuntu: ~/xv6-public
provides pointers to on-line resources for v6.
2000)). See also https://pdos.csail.mit.edu/6.828/, which
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 1
4,
xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (
Peer
ACKNOWLEDGMENTS
but is implemented for a modern x86-based multiprocessor using A
NSI C.
Version 6 (v6). xv6 loosely follows the structure and style of
v6,
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson'
s Unix
(https://github.com/mit-pdos/xv6-riscv.git)
our efforts to the RISC-V version
NOTE: we have stopped maintaining the x86 version of xv6, and sw
itched
$ htac 10.3 README
Type a valid integer between 1-1024
$
```

“htac n <filename>” 에서 n이 정수가 아니라면 경고문 출력 후 종료한다.

is_valid_integer(const char *str) 함수로 ‘0’, ‘9’포함 둘 사이의 char형만으로 구성되어 있는지 확인.

ex) htac 5.2 README, htac 23OS README, htac 5+1 README 실행결과

(5.2에서 ‘.’, 23OS에서 ‘O’, 5+1에서 ‘+’가 ‘0’, ‘9’포함 둘 사이의 char형이 아니므로 is_valid_ineger가 거짓임을 나타내는 ‘0’을 반환함으로 경고문을 출력한다.)

```
root@ubuntu: /home/taehyounkim/xv6-...
4,
xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (
Peer

ACKNOWLEDGMENTS

but is implemented for a modern x86-based multiprocessor using A
NSI C.
Version 6 (v6).  xv6 loosely follows the structure and style of
v6,
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson'
s Unix

(https://github.com/mit-pdos/xv6-riscv.git)
our efforts to the RISC-V version
NOTE: we have stopped maintaining the x86 version of xv6, and sw
itched
$ htac 5.2 README
Type a valid integer between 1-1024
$ htac 23OS README
Type a valid integer between 1-1024
$ htac 5+1 README
Type a valid integer between 1-1024
$ root@ubuntu:/home/taehyounkim/xv6-public#
```

“htac n <filename> <filename>”처럼 4개이상의 인자가 주어진다면 사용문 출력 후 종료한다.

ex)htac 5 README README2 (정상적으로 0,1,2번째 인자를 입력하였지만 그뒤로 4번째 인자를 입력하였으므로 명령어 형식을 지켜줄 것을 알리게 된다.)

htac -1 README README(이 경우 첫번째인자인 -1로 예외가 처리되지 않고 인자 3개를 입력하지 않는 것으로 예외처리 함)

```
root@ubuntu: /home/taehyounkim/xv6-public
raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ htac 5 README
simulator and run "make qemu".
Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC
x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
will need to install a cross-compiler gcc suite capable of producing
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
$ htac 5 README2
Can't open README2.
$ htac 5 README README2
Usage: htac <lines> <file name>
$ htac -1 README README
Usage: htac <lines> <file name>
$ htac 1.1 README
Type a valid integer between 1-1024
$ htac README 5
Type a valid integer between 1-1024
$
```

4. 수정한 소스코드(helloxv6.c, htac.c, Makefile)

helloxv6.c 소스코드

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc, char** argv)
{
    printf(1, "Hello xv6 World\n");
    exit();
}
```

htac.c 소스코드 (보고서에만 추가적으로 영문 주석 옆에 한글로 설명하였습니다./vi에서 한영 변경이 가능한 방법이 있다고 들었지만 영문 작성이 편하여 영어로 작성했습니다.)

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

```
char buf[512]; //just to follow cat.c buffer size (cat.c 파일과 버퍼 사이즈 일관성 유지)
char *lines_arr[1024]; //maximum 1024 lines can be stored (최대 1024줄만 출력가능)
int line = 0; //counts stored lines (저장된 줄 수, 출력 시 lines_arr의 인덱스로 쓰임)
```

```
int is_valid_integer(const char *str) { //(유효성 판단 정수인지 체크)
    while (*str) {
        if (*str < '0' || *str > '9') {
            return 0; //not valid integer
        }
        str++;
    }
    return 1; //valid integer
}
```

```
void htac(int fd) {
    int n; // number of bytes read in read() (읽어온 바이트 수)
    int total = 0; // total number of bytes currently in buf
    int last_newline = 0; // current line's starting index

    //Reading file
    while((n = read(fd, buf + total, sizeof(buf) - total - 1)) > 0) {
        //read data from file, store in buf start from the position 'buf+total'(not to overwrite data)
        //(파일에서 buf+total 위치부터 데이터를 읽어와 버퍼에 저장, 중복하여 데이터를 읽지 않기 위함)
    }
```



```

//process the data in current buf (현재 버퍼에 있는 데이터를 실행)
total += n; //total(+num of bytes read) (현재 버퍼에 담긴 바이트수 업데이트)
for(int i = 0; i < total; i++) { //iterate every character in buf(버퍼에 담긴 문자 순서대로 처리)
    if(buf[i] == '\n') { //check end of line (줄의 끝 지점(개행문자)일 경우)
        int length = buf + i - (buf + last_newline); //length of current line (현재 줄의 길이)
        lines_arr[line] = malloc(length + 1); //store line(allocate memory) (줄 저장, 메모리 할당)
        memmove(lines_arr[line], buf + last_newline, length);
        //copy line from buf to allocate memory (버퍼에서 할당된 메모리로 줄 복사)
        lines_arr[line][length] = 0; //add '\0' (줄 끝에 널 문자 추가)
        line++; //stored lines+1 (추가로 저장된 한 줄 추가)
        last_newline = i + 1; // next line's starting index (다음 줄의 시작 인덱스)

        if(line == 1024) { //since only maximum 1024 can be stored (최대 1024줄만 출력 가능
//함으로 1023줄이 넘는 파일의 경우 제일 처음 저장된 파일의 첫번째 줄 부터 메모리 할당을 해제하고, 나머지
//줄 한칸씩 앞으로 당김. 1줄은 마지막에 데이터가 여전히 남았을 경우에 추가하기 때문에 1024줄일때 첫줄을
//할당 해제하였음.)
            free(lines_arr[0]); //free first line
            for(int j = 0; j < 1023; j++) { //move every line up
                lines_arr[j] = lines_arr[j + 1];
            }
            line--;
        }
    }
}

//preparing buffer for next read (다음 읽을 부분으로 버퍼 채우기)
memmove(buf, buf + last_newline, total - last_newline); //move left data to start of buf
total -= last_newline; //left data
last_newline = 0; // reset last_newline(since buf is updated to left data)
}

if(total > 0) { // check unprocessed characters still left in buffer (마지막으로 여전히 버퍼에 남은
//데이터가 있을 경우 그 나머지 데이터 한 줄을 추가.)
    lines_arr[line] = malloc(total + 1); //allocate memory(store left characters in buf)
    memmove(lines_arr[line], buf, total); //copy left charters from buf to line_arr[line]
    lines_arr[line][total] = 0; //add '\0'
    line++;
}
}

int main(int argc, char *argv[]) {
    int n; //requested number of lines from user
    n = atoi(argv[1]); //convert argv[1] to integer(to check range) (예외체크를 위해 문자열을 정수형으로
//변환)

    if(argc!=3){ //check if it consists only 3 arguments(if not exit()) (인자가 3개가 아닌경우)

```

```

    printf(1, "Usage: htac <lines> <file name>\n");
    exit();
}
if (!is_valid_integer(argv[1])) { //check if it's valid integer(if not exit()) (1번째 인자(argv[1])가 정수가
//아닌경우)
    printf(1, "Type a valid integer between 1-1024\n");
    exit();
}

if (n > 1024 || n < 1) { //check if requested number is within range(if not exit()) (첫 번째인자가
//지정된 범위를 초과할 경우)
    printf(1, "Type between 1-1024\n");
    exit();
}
//Read file (모든 예외처리를 통과한 경우 파일 읽음)
int fd;
if((fd = open(argv[2], 0)) < 0){
    printf(1, "Can't open %s.\n", argv[2]);
    exit();
}
htac(fd);
close(fd);
//Determine the start index of line for printing(among lines_arr) (line_arr에 저장된 줄들 중 출력
//대상이 되는 줄들의 첫 번째 인덱스를 결정, 출력시에는 역순으로 첫번째 인덱스를 마지막으로 출력함)
int start_idx; //going to be last index of line to be printed
if (line - n >= 0) { //case)number of stored lines >= number of requested lines
    start_idx = line - n; // set the index line-n (line_arr에 저장된 줄 수가 첫번째 인자인 라인수보
//다 클 경우, 첫 번째인덱스를 line-n, 즉 저장된 줄수-첫번째인자(라인수)으로 설정)
} else { //(이외의 경우에는 저장된 모든 라인을 출력하기에 첫번째 인덱스를 0으로 설정)
    start_idx = 0; //set start index 0(will print all lines stored)
}

// Print the lines in reverse order (저장된 라인을 인덱스 line-1(마지막줄)부터 start_idx까지 역순으
//로 출력)
for(int i = line - 1; i >= start_idx; i--) {
    printf(1, "%s\n", lines_arr[i]);
}
// Free allocated memory (메모리 할당 해제 후 프로그램 종료)
for(int i = 0; i < line; i++) {
    free(lines_arr[i]); //free each lines
}
exit();
}

```

Makefile 수정 소스코드 (helloxv6, htac 구현 완료 후)

수정 내용 :

프로그램을 컴파일하기 위해 "helloxv6.c"와 "htac.c" 파일이 Makefile에서 인식되어야 한다. "UPROGS"에는 "_helloxv6"와 "_htac"를 추가 "EXTRA"에는 "helloxv6.c"와 "htac.c"를 추가한다.

```
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 $(CC) -dumpspeaks 2>/dev/null | grep -e '^[f]no-pie'),)
CFLAGS += -fno-pie -no-pie
endif
ifneq ($(shell $(CC) -dumpspeaks 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: $(OBS) 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.
```

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

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

```
$(LD) $(LDFLAGS) -T kernel.ld -o kernelmemfs entry.o $(MEMFSOBS) -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: %.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\  
README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\  
.gdbinit.tmpl gdbutil\  
helloxv6.c htac.c\
```

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