

과제 #1 : system calls in xv6

○ 과제 목표

- xv6 설치 및 컴파일
- Hello xv6 World" 출력하는 "helloxv6" 응용 프로그램을 위한 helloxv6.c 구현
- xv6에 lseek() 시스템 호출 추가 후 lseek()을 이용한 lseek_test.c 구현

○ 기본 지식

- xv6
 - ✓ 미국 MIT에서 멀티프로세서 x86 및 RISC-V 시스템을 위해 개발한 교육용 운영체제
 - ✓ UNIX V6를 ANSI C 기반으로 구현
 - ✓ 리눅스나 BSD와 달리 xv6은 단순하지만 UNIX 운영체제의 중요 개념과 구성을 포함하고 있음
- Cross Compile 방법 학습
 - ✓ xv6에는 텍스트 편집기 또는 gcc 컴파일러가 없음. 따라서 본 과제에서는 자신의 리눅스 환경에서 xv6 프로그램 작성 및 컴파일 후, 생성된 실행파일을 xv6 상에서 수행함
- 시스템 호출 추가 방법 이해
 - ✓ 기존 시스템 호출의 구현을 따라 새 시스템 호출을 추가하는 방법을 이해
 - 특정 시스템 호출은 인자가 없고 정수값만 리턴
 - ☞ 예 : sysproc.c 내 구현된 uptime()
 - 특정부 시스템 호출은 문자열 및 정수 등 여러 인수를 받아 간단한 정수 값을 리턴
 - ☞ 예 : sysfile.c 내 구현된 open()
 - 특정 시스템 호출은 여러 정보를 사용자가 정의한 구조체로 사용자 프로그램에 리턴
 - ☞ 예. fstat()은 파일에 대한 정보를 struct stat를 넣고 이 구조체를 가져와서 ls 응용 프로그램에 의해 파일에 대한 정보를 표준 출력
- Cross Compile 방법 학습
 - ✓ xv6에는 텍스트 편집기 또는 gcc 컴파일러가 없음. 따라서 자신의 리눅스 시스템에서 vi를 이용하여 프로그램 작성하고 컴파일 하고 나온 실행파일을 xv6 상에서 수행
- xv6 커널 이해
 - ✓ proc.c, proc.h, syscall.c, syscall.h, sysproc.c, user.h, usys.S 수정 필요
 - user.h : xv6의 시스템 호출 정의
 - usys.S : xv6의 시스템 호출 리스트
 - syscall.h : 시스템 호출 번호 매핑. -> 새 시스템 호출을 위해 새로운 매핑 추가
 - syscall.c : 시스템 호출 인수를 구문 분석하는 함수 및 실제 시스템 호출 구현에 대한 포인터
 - sysproc.c : 프로세스 관련 시스템 호출 구현. -> 여기에 시스템 호출 코드를 추가
 - proc.h는 struct proc 구조 정의 -> 프로세스에 대한 추가 정보를 추적을 위해 구조 변경
 - proc.c : 프로세스 간의 스케줄링 및 컨텍스트 전환을 수행하는 함수

○ 과제 내용

1. xv6 설치 및 컴파일

- ✓ xv6 다운로드

```
$ git clone https://github.com/mit-pdos/xv6-public
```

- ✓ QEMU 다운로드 및 설치

- xv6 운영체제는 자신의 컴퓨터에서 x86 하드웨어를 에뮬레이트 하는 QEMU x86 에뮬레이터에서 실행됨 (에뮬레이터 없이도 운용 가능하나, 수정을 위해 에뮬레이터 사용을 권장)

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

- ✓ xv6 컴파일 및 실행

```
$ make
$ make qemu
```

(예시 1). make qemu 실행 결과

```
root@oslab: /home/oslab/xv6-public
root@oslab:/home/oslab/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
$
```

(예시 2). ls 실행 결과

```
root@oslab: /home/oslab/xv6-public
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat        2 3 16244
echo       2 4 15100
forktest  2 5 9404
grep       2 6 18460
init       2 7 15680
kill       2 8 15128
ln         2 9 14980
ls         2 10 17612
mkdir      2 11 15224
rm         2 12 15204
sh         2 13 27844
stressfs   2 14 16116
usertests  2 15 67220
wc         2 16 16980
zombie     2 17 14792
console    3 18 0
```

2. helloxv6 응용 프로그램 구현

- "Hello xv6 World"를 출력하는 helloxv6.c 응용 프로그램 구현
- Makefile을 수정하여 helloxv6.c 파일도 make 시 컴파일 되도록 변경
- xv6 컴파일, 실행 후 helloxv6 응용 프로그램의 실행파일(명령어) 실행

(예시 3). helloxv6.c 파일 작성 예시

```
//helloxv6.c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(int argc, char **argv)
{
    printf(1, "Hello xv6 World\n");
    exit();
}
```

(예시 4). Makefile 수정 예시 1

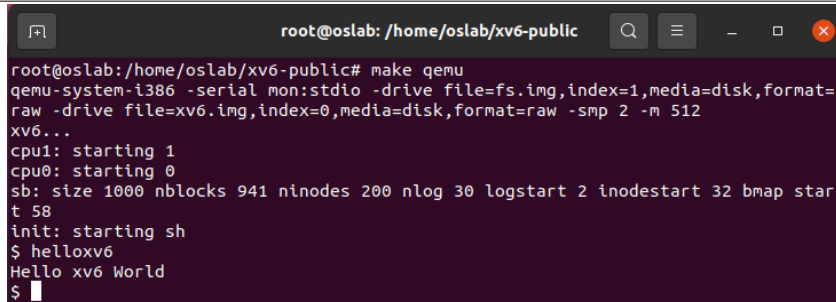
```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
```

```
_wc\  
_zombie\
```

(예시 5). Makefile 수정 예시 2

```
EXTRA=\n  
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 list.txt dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\  
.gdbinit.tmpl gdbutil\
```

(예시 6). helloxv6 실행 결과



```
root@oslab: /home/oslab/xv6-public  
root@oslab:/home/oslab/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 star  
t 58  
init: starting sh  
$ helloxv6  
Hello xv6 World  
$
```

3. lseek() 시스템 콜 추가

✓ 파일의 오프셋을 이동시키는 lseek() 시스템 호출 구현

- file.h의 struct file 사용

- lseek()의 프로토타입

☞ off_t lseek(int fd, off_t offset, int whence);

☞ 성공 시 오프셋, 실패시 -1을 리턴

☞ offset : 32bit integer로 0 또는 양의 정수만 가능, whence : SEEK_SET, SEEK_CUR, SEEK_END

- 파일 위치 포인터를 whence를 기준으로 offset만큼 이동

(예시 7). user.h에 lseek 추가

```
// system calls  
int fork(void);  
int exit(void) __attribute__((noreturn));  
int wait(void);  
int pipe(int*);  
int write(int, const void*, int);  
int read(int, void*, int);  
int close(int);  
int kill(int);  
int exec(char*, char**);  
int open(const char*, int);  
int mknod(const char*, short, short);  
int unlink(const char*);  
int fstat(int fd, struct stat*);  
int link(const char*, const char*);  
int mkdir(const char*);  
int chdir(const char*);  
int dup(int);  
int getpid(void);  
char* sbrk(int);  
int sleep(int);  
int uptime(void);  
...
```

(예시 8). usys.S에 lseek 추가

```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
    .globl name; \
    name: \
        movl $SYS_ ## name, %eax; \
        int $T_SYSCALL; \
        ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
...
```

(예시 9). syscall.h에 lseek 시스템 콜 번호 부여

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid  11
#define SYS_sbrk    12
#define SYS_sleep   13
#define SYS_uptime  14
#define SYS_open    15
#define SYS_write   16
#define SYS_mknod   17
#define SYS_unlink  18
#define SYS_link    19
#define SYS_mkdir   20
#define SYS_close   21
...
```

(예시 10). syscall.c에 lseek() 추가

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
```

```

extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
...

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]    sys_fstat,
[SYS_chdir]    sys_chdir,
[SYS_dup]      sys_dup,
[SYS_getpid]   sys_getpid,
[SYS_sbrk]     sys_sbrk,
[SYS_sleep]    sys_sleep,
[SYS_uptime]   sys_uptime,
[SYS_open]     sys_open,
[SYS_write]    sys_write,
[SYS_mknod]    sys_mknod,
[SYS_unlink]   sys_unlink,
[SYS_link]     sys_link,
[SYS_mkdir]    sys_mkdir,
[SYS_close]    sys_close,
...

```

(예시 11). lseek 시스템 콜 구현

```

//#include

int
sys_lseek(void)
{
    // SEEK_SET, SEEK_CUR, SEEK_END
}

```

4. lseek() 시스템 콜을 호출하는 간단한 셸 프로그램 구현

- (예시 12)에 따른 (예시 13)과 동일한 결과가 나올 수 있는 셸 프로그램 구현

(예시 12) hello.txt의 내용

Hello SSU

(예시 13). lseektest 실행 결과

```
$ lseektest hello.txt 6 World
Before : Hello SSU
After  : Hello World
```

(예시 14). lseektest 쉘 프로그램 구현

```
//lseektest.c
int main(int argc, char *argv[])
{
    int fd;
    if(argc < 4){
        printf(1, "usage : lseek_test <filename> <offset> <string>\n");
        exit();
    }
    // 기존 file 내용 출력
    // 변경 이후 file 내용 출력
    exit();
}
```

○ 과제 제출 마감

- 2024년 09월 29일 (일) 23시 59분까지 구글클래스룸으로 제출
- 보고서 (hwp, doc, docx 등으로 작성 - helloxv6 및 lseektest 프로그램이 수행된 결과 (캡처 등 포함)
- xv6에서 변경한 소스코드 및 테스트 쉘 프로그램 소스코드 (helloxv6.c, lseektest.c) 등
- 마감시간 이후 24시간까지 지연 제출 가능. 그 이후 제출은 0점 처리. 설계과제 마감시간 이후 지연 제출은 30% 감점.

○ 필수 구현(설치 및 설명 등)

- 1, 2, 3, 4

○ 배점 기준

- 1. xv6 설치 및 컴파일 : 30점
- 2. helloxv6 응용 프로그램 구현: 20점
- 3. lseek() 시스템 콜 추가 : 30점
- 4. lseektest 쉘 프로그램 구현 : 20점