과제 #1 : xv6 설치 및 system calls in xv6

○ 과제 목표
- xv6 설치 및 컴파일
- "Hello xv6 World" 출력하는 "helloxv6"와 프로세스의 상태를 출력하는 "psinfo" 응용프로그램을 위한 helloxv6.c 및 psinfo.c 구현
- xv6에 helloxv6(), get_procinfo() 등 두 개의 시스템 호출 추가 후, helloxv6()를 이용한 helloxv6.c get_procinfo()를 이용한 psinfo.c 구현하고 쉘에서 실행

○ 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 컴파일하고 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 (따라하기 수준. 설치과정도 보고서에 중간 과정을 캡쳐해서 제출해야 함.)
   - xv6 다운로드

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

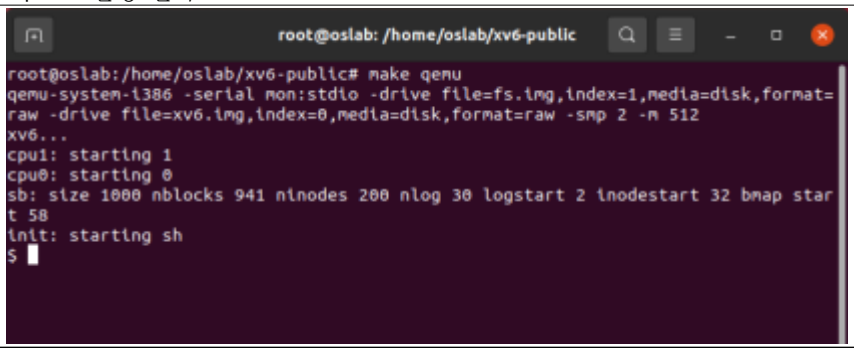
   - QEMU 다운로드 및 설치
     - ✓ xv6 운영체제는 자신의 컴퓨터에서 x86 하드웨어를 에뮬레이트 하는 QEMU x86 에뮬레이터에서 실행됨 (에뮬레이터 없이도 운용 가능하나, 수정을 위해 에뮬레이터 사용을 권장)

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

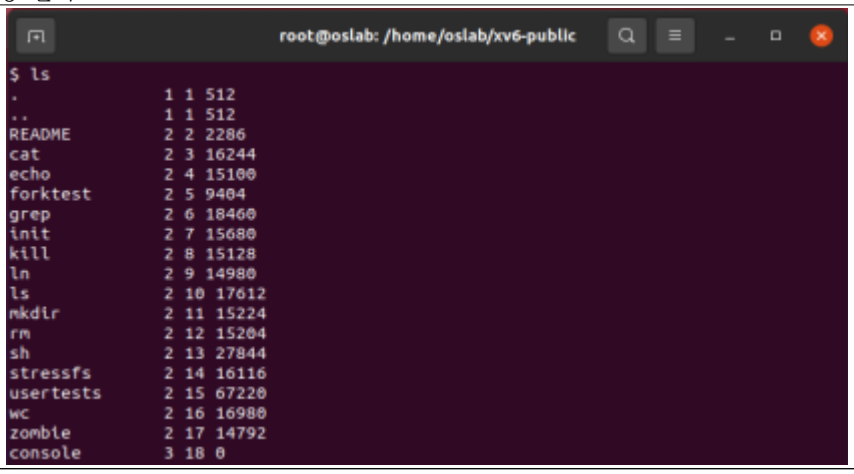
   - xv6 컴파일 및 실행

```
$ make
$ make qemu-nox
```

(예시 1). make qemu 실행 결과



(예시 2). ls 실행 결과



2. hello_number 시스템 콜 추가 (따라하기 수준. 보고서에 중간 과정을 캡쳐해서 제출해야 함.)

- hello_number 시스템 콜은 정수 하나를 인자로 받아 커널 내부에서 해당 정수를 출력 메시지에 포함하여 화면에 표시하고, 연산 결과를 호출한 프로세스에 리턴

✓ int hello_number(int n) : 인자 n을 받아, 커널 콘솔에 "Hello, xv6! Your number is n"이라는 메시지를 출력

✓ 이 때 n은 전달된 정수 값으로 치환되어 출력하고 n * 2 값을 반환하여 호출한 사용자 프로그램에 전달

✓ 입력으로 양수뿐 아니라 음수 등의 값 가능. 예. n=-7이면 리턴 값 -14. 리턴 시 사용 가능한 자료형 범위를 넘어서는 값은 입력으로 가정하지 않음 (xv6의 int는 32비트 정수).

- 시스템 콜 추가 절차 상세

✓ 새로운 시스템 콜 hello_number을 xv6에 추가하기 위해서는 다음과 같은 소스 파일들을 수정

✓ (1) syscall.h 수정: SYS_hello_number에 해당하는 시스템 콜 번호를 정의. xv6의 기존 시스템 콜 번호들은 SYS_fork (1)부터 순차적으로 할당되어 있음. 현재 xv6-public의 마지막 시스템 콜 번호 다음으로 새로운 번호를 지정. 예. 기존 SYS_close가 21이면 SYS_hello_number를 22로 정의). syscall.h에 다음과 같이 한 줄을 추가.

```
#define SYS_hello_number  22
```

✓ (2) sysproc.c 수정: 커널에서 실제로 동작할 시스템 콜 핸들러 함수를 구현. sysproc.c 파일에 새로운 함수 sys_hello_number(void)를 추가. 구현은 기존의 다른 시스템 콜을 참고하여 작성. 예. sys_kill이나 sys_sleep 함수처럼 argint() 함수를 이용해 인자 값을 커널 공간으로 가져올 수 있음

```
int
sys_hello_number(void) {
  int n;
  if(argint(0, &n) < 0)
    return -1;
  cprintf("Hello, xv6! Your number is %d\n", n);
  return n * 2;
}
```

☞ argint(0, &n)으로 사용자로부터 전달된 첫 번째 인자(n)를 읽어오고, xv6 커널의 cprintf 함수를 사용하여 커널 콘솔에 메시지를 출력.(cprintf는 커널에서 printf와 유사한 기능을 하며, defs.h에 프로토타입이 선언되어 있어 별도 구현하지 않아도 됨. xv6 커널

에서 바로 호출하여 사용.) 마지막으로 n*2 값을 반환하면, 이 값이 사용자 프로그램으로 전달

☞ cprintf로 출력하는 문구에는 개행 문자(₩n)를 넣어 줄바꿈을 해주는 것이 좋음. 또한 음수의 경우 %d 포맷으로 출력하면 부호까지 함께 출력됨.

✓ (3) syscall.c 수정: 시스템 콜 디스패처에 새로운 호출을 등록. syscall.c 파일에서 우선 새로 구현한 커널 함수를 extern으로 선언해야 함. 그리고 syscall 함수 포인터 배열에 해당 함수를 추가. 기존 배열에서 SYS_close 다음에 위치하도록 해야 함.

```
extern int sys_hello_number(void);
```

```
[SYS_close]    sys_close,
[SYS_hello_number] sys_hello_number,
```

☞ 이로써 커널이 트랩을 통해 hello_number 시스템 콜 번호(예: 22)를 받으면 sys_hello_number 함수를 호출하도록 연결됨

✓ (4) usys.S 수정: 사용자 영역에서 시스템 콜을 부를 수 있도록 어셈블리 코드 추가. usys.S 파일은 SYSCALL(name) 매크로를 이용해 각 시스템 콜의 사용자용 함수를 정의하고 있음. 마지막 라인에서 새로운 시스템 콜 이름을 넣어주면 되는데, 예를 들어 기존 SYSCALL(uptime) 다음에 추가.

```
SYSCALL(hello_number)
```

☞ helloxv6.c에서 hello_number(n)를 호출할 때, 어셈블리로 시스템 콜 트랩 (int $T_SYSCALL)이 발생하여 커널의 해당 시스템 콜로 연결

✓ (5) user.h 수정: 사용자 프로그램에서 새로운 시스템 콜 함수를 호출할 수 있도록 프로토타입을 선언. user.h 파일의 시스템 콜 함수 선언 리스트에 추가.

```
int hello_number(int n);
```

☞ 다른 시스템 콜들과 동일한 형식. 이 선언이 있어야 helloxv6.c를 컴파일할 때 링커가 extern 함수를 시스템 콜로 인식함.

3. get_procinfo 시스템 추가 (일부 학생들이 구현)

- get_procinfo 시스템 콜은 pid를 입력 받아 해당 process의 정보 출력. 인자가 없을 경우 자신의 pid

```
int get_procinfo(int pid, struct procinfo *uinfo);
// pid > 0 : 해당 PID 프로세스의 정보를 조회
// pid <= 0 : 호출한 자기 자신 정보 조회
// 리턴값: 성공 시 0, 실패 시 -1 (예: PID 없음, 포인터 오류 등)
```

- user.h와 sysproc.c에 동일한 구조체 정의

```
// user.h (유저용 선언)
struct procinfo {
  int  pid;            // 대상 PID
  int  ppid;           // 부모 PID
  int  state;          // enum procstate 값
  uint sz;             // 메모리 크기(바이트)
  char name[16];       // 프로세스 이름
};
int get_procinfo(int pid, struct procinfo *uinfo);
```

- 수정사항

✓ syscall.h : #define SYS_procinfo XX (빈 번호에 할당)

✓ usys.S : SYSCALL(procinfo) 한 줄 추가

✓ syscall.c: extern int sys_procinfo(void); 선언 후, syscalls[] 배열에 [SYS_procinfo] = sys_procinfo, 등록

✓ user.h : 위 프로토타입/구조체 선언 추가

- sysproc.c (일부)

```
// (커널 쪽 레이아웃도 동일하게 선언)
struct k_procinfo {
  int  pid, ppid, state; uint sz; char name[16];
};

int sys_get_procinfo(void) {
  int pid;
  char *uaddr;               // 유저 버퍼 시작 주소
  struct proc *p, *t;
  struct k_procinfo kinfo;

  if(argint(0, &pid) < 0) return -1;
  if(argptr(1, &uaddr, sizeof(struct k_procinfo)) < 0) return -1;

  acquire(&ptable.lock);
  if(pid <= 0) t = myproc();
  else {
    t = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
      if(p->pid == pid) { t = p; break; }
  }
  if(t == 0 || (t->state == UNUSED)) { release(&ptable.lock); return -1; }

  // 채우기 - 학생들이 직접 구현

  // 유저 공간으로 복사
  if(copyout(myproc()->pgdir, (uint)uaddr, (void*)&kinfo, sizeof(kinfo)) < 0)
    return -1;
  return 0;
}
```

4. 사용자 응용 프로그램 작성과 쉘에서 테스트
- 추가한 새로운 시스템 콜을 테스트하기 위한 사용자 프로그램 helloxv6.c와 psinfo.c를 작성
- (1) helloxv6.c
  - ✓ 이 프로그램은 실행 시 커널의 hello_number 시스템 콜을 호출하고, 커널에서 출력한 메시지를 확인한 후 시스템 콜이 리턴한 값을 사용자 프로그램 측에서 출력
  - ✓ helloxv6.c의 main 함수에서 hello_number(5)를 호출하고, 함수의 반환값을 변수로 받아 저장. 이후 사용자 영역에서 printf를 사용하여 반환된 값을 출력합니다. 또한, 필요하다면 hello_number(-7)과 같은 호출을 추가로 수행하여 음수 인자에 대해서도 올바른 결과가 출력되는지 확인

```
// helloxv6.c (사용자 프로그램 예시)
#include "types.h"
#include "stat.h"
#include "user.h"

int main(int argc, char *argv[]) {
  int res = hello_number(5);
  printf(1, "hello_number(5) returned %d\n", res);
  // 추가 테스트: 음수 인자 호출 예시
  // int res2 = hello_number(-7);
  // printf(1, "hello_number(-7) returned %d\n", res2);
  exit();
}
```

- (2) psinfo.c

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

static char* s2str(int s){
  switch(s){
    case 0: return "UNUSED"; case 1: return "EMBRYO";
    case 2: return "SLEEPING"; case 3: return "RUNNABLE";
    case 4: return "RUNNING"; case 5: return "ZOMBIE";
  } return "UNKNOWN";
}

int main(int argc, char *argv[]){
  struct procinfo info;
  int pid = (argc >= 2) ? atoi(argv[1]) : 0; // 0이면 자기 자신
  // 채우기 - 학생들이 직접 구현
  printf(1, "PID=%d PPID=%d STATE=%s SZ=%d NAME=%s\n",
         info.pid, info.ppid, s2str(info.state), info.sz, info.name);
  exit();
}
```

- (3) Makefile 등록: 작성한 helloxv6.c와 psinfo.c를 xv6의 사용자 프로그램으로 포함시키기 위해, xv6 Makefile의 사용자 프로그램 목록에 해당 파일을 추가. Makefile의 UPROGS 변수에 다른 프로그램들과 동일한 형식으로 _helloxv6와 psinfo를 추가 (밑줄 _ 붙은 이름으로). 이를 통해 make 시 helloxv6.c와 psinfo.c가 컴파일되어 xv6 파일시스템 이미지에 포함됨
- (4) 실행 테스트: xv6를 재빌드한 후 QEMU로 부팅하여 xv6 셸에서 helloxv6 및 psinfo 명령을 실행.

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ helloxv6
Hello, xv6! Your number is 5
hello_number(5) returned 10
$ psinfo
PID=4 PPID=2 STATE=RUNNING SZ=12288 NAME=psinfo
$ psinfo  1
PID=1 PPID=0 STATE=SLEEPING SZ=12288 NAME=init
$ psinfo 9999
psinfo: failed (pid=9999)
$
```

○ 과제 제출 마감
- 2025년 9월 21일(일) 23시 59분까지 구글클래스로 제출 (강의계획서 확인). 1초라도 늦으면 0점 처리
- 가산점 2 부여 마감 기한 : 2025년 9월 14일(일) 23시 59분까지 구글클래스로 제출 1초라도 늦으면 가산점2 없음

○ 주의사항
- 보고서 (hwp, doc, docx 등으로 작성 - 수행된 결과 (캡쳐 등) 반드시 포함시킬 것
- xv6에서 변경한 소스코드 및 테스트 쉘 프로그램 소스코드 (helloxv6.c, psinfo.c) 등
- 보고서에는 본 설계명세서에서 주어진 소스코드의 분석도 반드시 포함되어야 함

○ 필수 구현
- 1, 2, 3, 4 (1의 xv6 설치 과정은 화면 캡쳐 등을 이용하여 상세하게 설명한 것을 보고서에만 포함시키면 됨)

○ 배점 기준
- 1. xv6 설치 및 컴파일 : 10점
- 2. helloxv62 시스템 콜 추가 : 20점
- 3. procinfo 시스템 콜 추가 : 40점
- 4. helloxv6, psinfo 쉘 프로그램 구현 및 테스트 : 30점