과제 #2 : XV6에 System Call 추가

○ 과제 목표

- xv6에 새로운 시스템 호출 추가

○ 기본 지식

- 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. memsize() 시스템 콜 추가 및 이를 호출하는 간단한 쉘 프로그램 구현

✓ 호출한 프로세스의 메모리 사용량을 출력하는 memsize() 시스템 호출 구현

- xv6 수정을 통한 시스템 호출 추가
- 커널 모드의 myproc() 사용

(예시 1). memsize 시스템 콜 구현

```
int
sys_memsize(void)
{
        uint size;
        //기능 구현
        return size;
}
```

(예시 2). proc.c의 myproc 함수 확인

```
struct proc*
myproc(void) {
        struct cpu *c;
        struct proc *p;
        pushcli();
        c = mycpu();
        p = c->proc;
        popcli();
        return p;
}
```

(예시 3). proc.h 확인

```
struct proc {
        uint sz;                        // Size of process memory (bytes)
        pde_t* pgdir;                   // Page table
        char *kstack;                   // Bottom of kernel stack for this process
        enum procstate state;           // Process state
        int pid;                        // Process ID
        struct proc *parent;            // Parent process
        struct trapframe *tf;           // Trap frame for current syscall
        struct context *context;    // swtch() here to run process
        void *chan;                     // If non-zero, sleeping on chan
        int killed;                 // If non-zero, have been killed
        struct file *ofile[NOFILE];  // Open files
        struct inode *cwd;              // Current directory
        char name[16];                  // Process name (debugging)
};
```

✓ memsize() 실행 확인

- memsizetest 쉘 프로그램 구현
- 아래 예시와 동일한 결과가 나와야 함

(예시 4). memsizetest 쉘 프로그램 구현 예시

```
#define SIZE 2048

int main(void)
{
        int msize = memsize();
        printf(1, "The process is using %dB\n", msize);

        char *tmp = (char *)malloc(SIZE * sizeof(char));

        printf(1, "Allocating more memory\n");
        msize = memsize();
        printf(1, "The process is using %dB\n", msize);

        free(tmp);
        printf(1, "Freeing memory\n");
        msize = memsize();
        printf(1, "The process is using %dB\n", msize);

        exit();
}
```

(예시 5). memsizetest 실행 결과

```
$ memsizetest
The process is using 12288B
Allocating more memory
The process is using 45056B
Freeing memory
The process is using 45056B
```

✓ 위 실행 결과에서 malloc 전, 후 차이가 2048 바이트가 아닌 이유를 설명
- sz의 단위가 바이트(byte)가 아닌 비트(bit)
- 8로 나눌 경우(bit to byte) 2048이 아닌 4096 바이트가 나옴
- malloc.c를 분석하여 4096 바이트가 나오는 이유를 설명

2. trace 시스템 콜 추가 및 이를 호출하는 간단한 쉘 프로그램

✓ 이후 과제를 디버깅할 때 도움이 될 수 있는 trace 시스템 콜 구현
- 추적할 시스템 콜을 지정하는 정수 [mask]를 인자로 받음
  ☞ ex). read 시스템 콜을 추적하기 위해 사용자 프로그램은 trace(1 << SYS_read)을 호출함
  ☞ SYS_read는 syscall.h 에 정의되어있는 시스템 콜 번호
- xv6 커널을 수정하여 시스템 콜 번호가 마스크에 설정되어 있는 경우 각 시스템 콜이 리턴될 때 프로세스 아이디, 시스템 콜 이름, 리턴 값이 출력되어야 함
  ☞ 출력 형식 : "syscall traced: pid = %d, syscall = %s, %d returned\n"
- trace 시스템 콜은 호출한 프로세스와 호출 이후 생성(fork)하는 모든 자식 프로세스에 대한 trace mask를 활성화 해야 하고, 다른 프로세스에는 영향을 미치면 안됨
- 아래 예시 외에도 xv6 커널의 추가적인 수정이 필요함

(예시 6). proc 구조체 수정 필요

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

(예시 7). sysproc.c에 sys_trace 구현

```
int
sys_trace(void)
{


}
```

(예시 8). 자식 프로세스 생성 시 mask를 전달하기 위해 fork 수정

```
int
fork(void)
{


}
```

✓ trace 시스템 콜 실행 확인

- ssu_trace 쉘 프로그램 구현
- ssu_trace
- $ ssu_trace [mask] [command]
  - ☞ 위에서 설명한 것과 같이 [mask]는 시스템 콜을 추적하기 위한 비트 집합
  - ☞ exec을 통해 [command]를 실행하여, [command]가 호출하는 시스템 콜을 추적

32는 2의 5승. 5번을 trace 해라.

(예시 9). ssu_trace 실행 예시

```
$ ssu_trace 32 grep ssu_os README
syscall traced: pid = 4, syscall = read, 1023 returned
syscall traced: pid = 4, syscall = read, 988 returned
syscall traced: pid = 4, syscall = read, 275 returned
syscall traced: pid = 4, syscall = read, 0 returned

$ ssu_trace 2130080 grep ssu_os README
syscall traced: pid = 13, syscall = exec, 0 returned
syscall traced: pid = 13, syscall = open, 3 returned
syscall traced: pid = 13, syscall = read, 1023 returned
syscall traced: pid = 13, syscall = read, 988 returned
syscall traced: pid = 13, syscall = read, 275 returned
syscall traced: pid = 13, syscall = read, 0 returned
syscall traced: pid = 13, syscall = close, 0 returned
```

Ex) 21300이 2의 8승+2의 5승 -> 8번 5번 trace

○ 과제 제출 마감

- 2022년 09월 21일 (수) 23시 59분 59초까지 구글클래스룸으로 제출
- 보고서 (hwp, doc, docx 등으로 작성 - 총 2개의 테스트 프로그램이 수행된 결과 (캡쳐 등 포함)
- xv6에서 변경한 소스코드 및 테스트 쉘 프로그램 2개(memsizetest, ssu_trace)
- 1일 지연 제출마다 30% 감점. 4일 지연 제출 시 0점 처리 (이하 모든 설계 과제 동일하게 적용)

○ 필수 구현(설치 및 설명 등)

- 1, 2

○ 배점 기준

- 1 : 50점
- 2 : 50점