REPORT

- 제 목 : Project1 Wiki
- 수강과목 : 2024HY13065_운영체제
- 담당 교수: 강수용 교수님
- 학 과 : 컴퓨터소프트웨어학부
- 학 번 : 2019089270
- 이 름 : 김주호
- 제출일자: 2024.03.30

Outline

- getgpid() 시스템 콜 구현
  - 부모 프로세스의 부모프로세스의 pid를 반환하는 시스템 콜로 유저 프로그램에서 사용이 가능한 형태이다.
- project01 User program 구현
  - getgpid()를 이용하여 아래와 같은 결과를 화면에 출력할 수 있도록하는 유저프로그램이다.

```
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
$ project01
My student id is 2019089270
My pid is 3
My gpid is 1
$
```

Design

시스템 콜 추가 과정

커널 영역

1. 함수 정의
    a. ex_syscall.c 파일에 getgpid 함수 정의
2. makefile에 syscall 파일 추가
3. defs.h에 정의 추가
    a. defs.h에 getgpid 선언
4. wrapper function 구현
    a. ex_syscall.c에 sys_getgpid 함수 정의
5. wrapper function을 새로운 syscall로 등록
    a. syscall.h 와 syscall.c 에 wrapper function 추가

유저 영역

1. syscall을 유저가 볼 수 있도록 처리
    a. user.h 파일에 getgpid 추가
2. 유저가 syscall을 사용할때 어떻게 전처리할지 처리
    a. usys.S에 macro 추가

getgpid() 함수 구현

1. 프로세스 구조체 활용
    a. 과제 디렉토리 내부를 보면 xv6 os가 관리하는 프로세스 구조체를 찾을 수 있는데 해당 구조체의 pid와 parent를 이용할 것이다.

```
//proc.h
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)
};
```

2. 현재 프로세스 구조체를 가져올 수 있는 function 활용
    a. 과제 디렉토리 내부에 myproc function을 이용하여 현재 프로세스 구조체의 포인트를 가져올 수 있도록한다.

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

3. 위의 2가지를 조합하여 getgpid 함수를 구현할 것이다.

User Program 구현 계획

1. 구현된 getgpid 함수 이용
2. 내장되어있는 getpid 함수를 이용
3. 구현된 user program을 makefile에 추가

Implement

getgpid syscall 추가

1. xv6-public/ex_syscall.c 파일 생성 및 getgpid 추가
    a. 현재 프로세스의 부모의 부모의 프로세스 아이디를 찾는 형태로 구현하였다.

```
#include "types.h" //없으면 defs.h에서 필요한 types가 존재하지 않아 실행되지 않음
#include "mmu.h" //defs를 위한 parameter1
#include "param.h" //defs를 위한 parameter2
#include "defs.h"
#include "proc.h" //proc.h에서 myproc()를 사용하기 때문에 include


int getgpid(void){
    struct proc *p = myproc();
    struct proc *pp = p->parent;
    struct proc *gpp= pp->parent;
    return gpp->pid;
}
```

2. xv6-public/MakeFile에 ex_syscall.o 추가
    a. makefile에서 작성한 ex_syscall를 찾을 수 있도록 수정해준다.

```
    OBJS = \
        ...
        trapasm.o\
        trap.o\
        uart.o\
        vectors.o\
        vm.o\
        ex_syscall.o\
```

3. xv6-public/defs.h에 getgpid 함수의 정의 추가
    a. 다른 c 파일들에게도 함수가 보여지기 위해서 정의를 추가한다.

```
...
void            clearpteu(pde_t *pgdir, char *uva);

//ex_syscall.c
int            getgpid(void);

// number of elements in fixed-size array
#define NELEM(x) (sizeof(x)/sizeof((x)[0]))
```

4. xv6-public/ex_syscall.c에 sys_getgpid 함수 정의를 추가한다.
    a. 현재는 wrapper함수가 딱히 하는 기능이 없지만 예외처리와 같은 추후 확장적 설계를 위해서 미리 구현해둔다.

```
...
    return gpp->pid;
}

int sys_getgpid(void){
    return getgpid();
}
```

5. xv6-public/syscall.h 와 xv6-public/syscall.c 에 sys_getgpid 추가
    a. wrapper function을 system call로써 사용하기 위해서 등록해준다.

```
//syscall.c
...
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getgpid(void); //sys_getgpid()를 추가
```

```
//syscall.h
...
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_getgpid 22
```

6. xv6-public/user.h에 getgpid 함수 추가
    a. 유저가 syscall을 볼 수 있도록 처리해주는 과정이다.

```
...
int sleep(int);
int uptime(void);
int getgpid(void);

// ulib.c
int stat(const char*, struct stat*);
```

7. xv6-public/usys.S에 getgpid 매크로 추가
    a. 유저 프로그램에서 syscall를 호출할때 전처리 과정을 위해서 추가해주는 과정이다.

```
...
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getgpid)
```

User Program 구현

1. xv6-public/project01.c 추가 및 작성
    a. user.h에 있는 getpid와 등록한 getgpid를 이용하여 원하는 화면에 출력하도록하는 함수 작성한다.

```
#include "types.h" //user.h에 필요한 타입이 있음
#include "user.h"

int main(int argc, char* argv[]){
    const int school_id=2019089270;
    printf(1,"My student id is %d\n",school_id);
    printf(1,"My pid is %d\n",getpid());
    printf(1,"My gpid is %d\n",getgpid());
    exit();
}
```

2. xv6-public/Makefile 수정
    a. user program이 같이 컴파일 될 수 있도록 Makefile 수정한다.

```
...
UPROGS=\
    ...
    _wc\
    _zombie\
    _project01\
...
EXTRA=\
    ...
```

```
    printf.c umalloc.c project01.c\
    ...
```

Result

OS 이미지 생성과정

1. make clean
    a. 혹시 모를 과거 데이터가 존재할 수 있으므로 컴파일된 파일들을 모두 삭제합니다.
2. make
    a. 추가된 ex_syscall.c와 함께 수정된 os의 파일들을 컴파일합니다.

```
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
ld -m    elf_i386 -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
boot block is 467 bytes (max 510)
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o swtch.o swtch.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o trapasm.o trapasm.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
./vectors.pl > vectors.S
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o vectors.o vectors.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o entry.o entry.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
ld -m    elf_i386 -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
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-poi
ld -m    elf_i386 -N -e start -Ttext 0 -o initcode.out initcode.o
objcopy -S -O binary initcode.out initcode
objdump -S initcode.o > initcode.asm
ld -m    elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0515362 s, 99.3 MB/s
```

```
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000102393 s, 5.0 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
420+1 records in
420+1 records out
215408 bytes (215 kB, 210 KiB) copied, 0.00144732 s, 149 MB/s
```

2. make fs.img
    a. project01.c를 포함한 실행파일들을 컴파일하고 운영체제에 실행파일들을 넣어 이미지를 생성합니다.

```
gcc -Werror -Wall -o mkfs mkfs.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o usys.o usys.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _cat cat.o ulib.o usys.o printf.o umalloc.o
objdump -S _cat > cat.asm
objdump -t _cat | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > cat.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _echo echo.o ulib.o usys.o printf.o umalloc.o
objdump -S _echo > echo.asm
objdump -t _echo | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > echo.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
# forktest has less library code linked in - needs to be small
# in order to be able to max out the proc table.
ld -m    elf_i386 -N -e main -Ttext 0 -o _forktest forktest.o ulib.o usys.o
objdump -S _forktest > forktest.asm
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _grep grep.o ulib.o usys.o printf.o umalloc.o
objdump -S _grep > grep.asm
objdump -t _grep | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > grep.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _init init.o ulib.o usys.o printf.o umalloc.o
objdump -S _init > init.asm
objdump -t _init | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > init.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _kill kill.o ulib.o usys.o printf.o umalloc.o
objdump -S _kill > kill.asm
objdump -t _kill | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kill.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _ln ln.o ulib.o usys.o printf.o umalloc.o
objdump -S _ln > ln.asm
objdump -t _ln | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ln.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _ls ls.o ulib.o usys.o printf.o umalloc.o
objdump -S _ls > ls.asm
objdump -t _ls | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ls.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _mkdir mkdir.o ulib.o usys.o printf.o umalloc.o
objdump -S _mkdir > mkdir.asm
objdump -t _mkdir | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > mkdir.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _rm rm.o ulib.o usys.o printf.o umalloc.o
objdump -S _rm > rm.asm
objdump -t _rm | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > rm.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _sh sh.o ulib.o usys.o printf.o umalloc.o
objdump -S _sh > sh.asm
objdump -t _sh | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > sh.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _stressfs stressfs.o ulib.o usys.o printf.o umalloc.o
```

```
objdump -S _stressfs > stressfs.asm
objdump -t _stressfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > stressfs.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _usertests usertests.o ulib.o usys.o printf.o umalloc.o
objdump -S _usertests > usertests.asm
objdump -t _usertests | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > usertests.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _wc wc.o ulib.o usys.o printf.o umalloc.o
objdump -S _wc > wc.asm
objdump -t _wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > wc.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf.o umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-po
ld -m    elf_i386 -N -e main -Ttext 0 -o _project01 project01.o ulib.o usys.o printf.o umalloc.o
objdump -S _project01 > project01.asm
objdump -t _project01 | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > project01.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _w
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 697 blocks have been allocated
balloc: write bitmap block at sector 58
```

실행과정

1. xv6-public/start.sh 작성 및 권한 부여

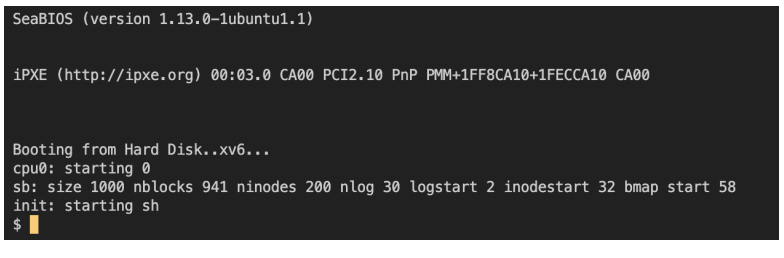
```
# !/bin/bash
qemu-system-i386 -nographic -serial mon:stdio -hdb fs.img xv6.img -smp 1 -m 512
```

(xv6-public의 디렉토리로 이동한 후)

```
$ sudo chmod 755 ./start.sh
```

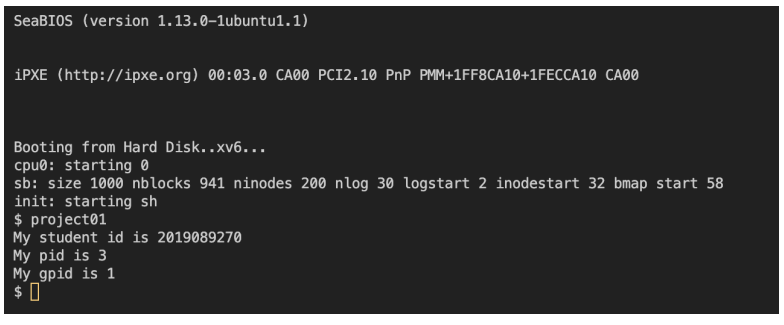
2. xv6 실행

```
$ ./start.sh
```

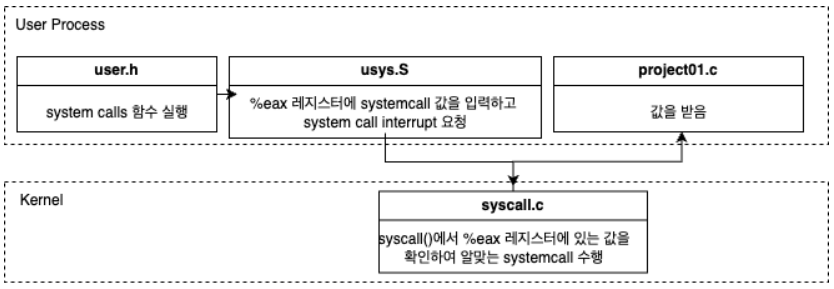


3. 유저 프로그램 실행

```
$ project01
```



유저 프로그램의 system call 실행 과정



1. user.h

```
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getgpid(void);
```

2. usys.S

```
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getgpid)
```

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

eax 레지스터에 값 매크로값 지정후 syscall interrupt 수행한다.

3. syscall.c

```
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,
[SYS_getgpid] sys_getgpid //sys_getgpid()를 추가
};
```

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

eax 레지스터에 있는 값을 확인후 지정된 system call 수행하고 결과를 eax 레지스터에 반환한다.

4. project01.c
   a. 최종적으로 2에서 실행된 함수가 끝나고 결과값이 eax에 있으므로 system call의 결과를 받아볼 수 있게된다.

Trouble Shooting

1. 많은 파일로 인한 적절히 써야하는 함수와 구조체를 찾는데 어려움이 있었다.
    a. Solution
        i. cscope를 이용하여 가능한 조합을 최대한 많이 검색해본다.

```
:cs find g getpid
:cs find s getpid
:cs find d getpid
```

2. 유저프로그램 작성시 프로세스가 끝나지 않는 문제가 발생했었다
    a. project01.c
        i. 수정전

```
#include "types.h" //user.h에 필요한 타입이 있음
#include "user.h"

int main(int argc, char* argv[]){
    const int school_id=2019089270;
    printf(1,"My student id is %d\n",school_id);
    printf(1,"My pid is %d\n",getpid());
    printf(1,"My gpid is %d\n",getgpid());
    return 0; //문제가 되는 부분
}
```

위와 같이 일반적인 C코드의 형태로 작성하면 원하는 것이 출력은 되나 프로세스가 정상적으로 종료되지 않아 trap 관련 로그가 남게 된다.

        ii. 수정후

```
#include "types.h" //user.h에 필요한 타입이 있음
#include "user.h"

int main(int argc, char* argv[]){
    const int school_id=2019089270;
    printf(1,"My student id is %d\n",school_id);
    printf(1,"My pid is %d\n",getpid());
    printf(1,"My gpid is %d\n",getgpid());
    exit();
}
```