Operating System Project01 Wiki

요구조건

- 부모 프로세스의 부모 프로세스(조부모)의 pid(process id)를 반환하는 시스템 콜인 getgpid() (get grand parent process id)를 구현
- getgpid() 시스템 콜은 유저 프로그램에서 사용될 수 있어야 함.
- 학번, 프로세스의 id, 조부모 프로세스의 id를 출력하는 유저 프로그램 “project01” 구현

Design

getgpid() 구현 계획 고민

`getgpid`를 어떻게 구현해야되나 고민하던 중 다른 시스템콜 함수들이 어떻게 구현되어있는지 코드를 보고 참고해보고자 여러 시스템 콜의 코드를 찾아보았다. xv6의 구조에 익숙하지

않아 헤매던 중 lab02에서 살펴보았던 `syscall.c`에서 `int sys_getpid(void)`가 선언되어 있는 것을 보고 프로세스의 id를 출력해주는 함수는 제공된다는 것을 알게 되었다.

`vim syscall.c`

```
#include "types.h"
#include "defs.h"
...
extern int sys_getpid(void)
```

`int sys_getpid(void)`가 구현된 코드가 있는 곳을 살펴보고자 여러 파일을 뒤져보았고 `sysproc.c` 파일 내에 구현되어있는걸 찾을 수 있었다.

`vim sysproc.c`

```
int
sys_getpid(void)
{
    return myproc()->pid;
}
```

여기서 `myproc()`은 `proc.c`에서 구현된 코드를 살펴본 결과 `struct proc*`를 반환한다는 걸 알 수 있었다. `struct proc`의 구조는 `proc.h` 파일 안에서 확인할 수 있었다.

`vim proc.h`

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

```
  char name[16];               // Process name (debugging)
};
```

sys_getpid(void)는 myproc()을 통해 현재 프로세스의 구조 내 pid에 접근하여 프로세스 id를 출력해주는 것을 알 수 있었다. 또한 struct proc 내에는 struct proc *parent, 즉 자신의 부모 프로세스의 정보를 담고 있기에 이를 활용하여 getgpid를 구현하기로 계획하였다.

getgpid 구현 방식 구상

lab02에서 새로운 시스템콜을 작성하는 방법을 참고하여 아래와 같은 순서로 구현을 하였다.

1. 새로운 소스코드 파일을 만들고 새로운 시스템 콜 `getgpid` 와 wrapper function을 커널에 구현한다.
2. 새로운 소스코드 파일을 만들었으니 `Makefile`에 object 파일을 추가한다
3. 새로운 시스템 콜을 `defs.h`, `syscall.h`, `syscall.c`에 추가해준다.

위의 과정을 거쳐 getgpid를 커널에 구현하고 이를 실행할 유저 프로그램 또한 구현한다.

1. 유저 프로그램 `project01.c`를 만든다
2. `user.h`에 `getgpid`를 추가하고 `usys.S`에 새로운 매크로를 추가한다.

Implement

prac_syscall.c

`prac_syscall.c` 라는 새로운 파일을 작성하여 `int getgpid(void)`를 구현하고 wrapper function인 `int sys_getgpid(void)`를 구현하였다.

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
```

```
int getgpid(void){

        return myproc()->parent->parent->pid;
}

int sys_getgpid(void){

        return getgpid();

}
```

앞서 언급한 `int getpid(void)`의 형태를 참고하여 myproc()의 parent의 parent에 접근하여 pid를 가져오는 방법으로 함수를 구현하였다. 여기서 include 한 헤더 파일들은 각각의 파일들이 아직 어떤 역할을 하는지 알지 못해서 일단 비슷한 기능을 수행하는 `int getpid(void)` 가 정의된 `sysproc.c` 내에 include 된 헤더파일을 똑같이 include 해주었다. 나중에 차근차근 살펴봐야할 것 같다.

Makefile

`prac_syscall.c`라는 새로운 파일을 작성하였으니 `Makefile` 내에 `prac_syscall.o` 파일을 추가해주어야 한다.

```
OBFS = \
    bio.o\
    cosole.o\
    ...
    vm.o\
    prac_syscall.o\
```

이후 `make | grep prac_syscall` 을 해주어야 성공적으로 object파일이 추가된다.

defs.h

새로 만든 시스템 콜을 다른 c 파일에서도 접근할 수 있도록 `defs.h`에 선언을 해주었다.

```
struct buf;
struct context;
...
```

```
int copyout(pde_t*, uint, void*, uint);
void clearpteu(pde_t *pgdir, char *uva);

//prac_syscall.c
int getgpid(void);
```

syscall.h

구현한 함수를 시스템 콜로 규정하기 위해 `syscall.h`에 sys_getgpid에 해당하는 시스템 콜 넘버를 정의하였다.

```
// System call numbers
#define SYS_fork 1
#define SYS_exit 2
...
#define SYS_close 21
#define SYS_getgpid 22
```

syscall.c

`syscall.c` 내에선 `int sys_getgpid(void)`를 선언해주고 syscall 내에서 레지스터에 넘겨주기 위한 배열에 추가하였다.

```
#include "types.h"
#include "defs.h"
...
extern int sys_uptime(void);
extern int sys_getgpid(void);
...
[SYS_close] sys_close,
[SYS_getgpid] sys_getgpid,
};
```

user.h

이후 작성할 유저프로그램에서 `getgpid()` 를 호출할 수 있도록 `user.h`에 추가해주었다.

```
struct stat;
struct rtcdate;

// system calls
int fork(void);
...
int uptime(void);
int getgpid(void);
```

usys.S

`usys.S`에도 똑같이 `getgpid()`에 대한 새로운 매크로를 추가해주었다.

```
#include "syscall.h"
#include "trap.h"

#define SYSCALL(name)\
...
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getgpid)
```

project01.c

새로운 시스템 콜을 실행할 유저프로그램 `project01.c`를 생성하였다. `project01.c` 내에서는 과제 명세에 맞게 학번을 출력하고 `getgpid()`, `getpid()`를 호출해 프로세스의 id와 조부모 프로세스의 id를 출력해주었다.

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

int main(int argc, char *argv[]){
        int pid=getpid();
        int gpid=getgpid();

        printf(1,"MY student id is 2019073309\n");
```

```
        printf(1,"My pid is %d\n", pid);
        printf(1,"My gpid is %d\n", gpid);

        exit();
}
```

`getpid()`와 `getgpid()` 두 함수 모두 `user.h`에 선언되어있기에 이를 사용하고자 `user.h`를 include 해주었다.

Makefile

새로운 유저프로그램을 `Makefile`에 추가해주었다.

```
...
UPROGS=\
        _cat\
        _echo\
        ...
        _zombie\
        _project01\
...
EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c k
    ln.c ls.c mkdir.c rm.c stressfs.c wc.c zombie.c\
    printf.c umalloc.c project01.c\
```

Result

터미널 상에서 xv6-public 디렉토리로 이동한다.

`cd xv6-public`

Makefile을 수정하여 `prac_syscall.o`를 추가해줬으므로 make | grep prac_syscall을 해주어야 한다.

`make clean`

`make | grep prac_syscall`

이후 fs.img를 만들어주고

`make fs.img`

xv6를 부팅한다.

```
qemu-system-i386 -nographic -serial mon:stdio -hdb fs.img xv6.img -smp 1 -m 512
```

```
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 2019073309
My pid is 3
My gpid is 1
$
```

xv6를 부팅하여 `project01`을 실행하면 정상적으로 학번과 pid, gpid가 출력되는걸 확인할 수 있다.

Trouble shooting

헤더파일 오류

`prac_syscall.c`를 작성하고 make를 하는 과정에서 자꾸 오류가 발생하였다.

```
In file included from prac_syscall.c:2:
proc.h:5:20: error: field 'ts' has incomplete type
    5 |   struct taskstate ts;         // Used by x86 to find stack for interrupt
      |                    ^~
proc.h:6:22: error: 'NSEGS' undeclared here (not in a function)
    6 |   struct segdesc gdt[NSEGS];   // x86 global descriptor table
      |                      ^~~~~
proc.h:13:24: error: 'NCPU' undeclared here (not in a function)
   13 | extern struct cpu cpus[NCPU];
      |                        ^~~~
proc.h:49:22: error: 'NOFILE' undeclared here (not in a function)
   49 |   struct file *ofile[NOFILE];  // Open files
      |                      ^~~~~~
```

에러 메세지상으로는 `proc.h`에 문제가 있는 것으로 보이는데 그럴 수가 없는게 교육용으로 만들어진 운영체제이고 이미 온라인 상에서 많이 배포가 된 코드인데 문제가 있다는게 말이 안된다. 또한 다른 소스파일에서도 버젓이 proc.h가 include 되어있는데 prac_syscall.c 파일을 제거하면 잘 작동이 되기 때문이다. 혼자서 고민하다 답을 찾을 수 없어서 주변 지인에게 자문을 한 결과 헤더파일을 include 하는 순서가 문제였다는 것을 알게 되었다.

```
#include "types.h"
#include "proc.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
```

이런식으로 proc.h에서 쓰이는 변수나 자료형이 다른 헤더파일에 정의가 되어있는데 proc.h를 먼저 include 해버리면 다른 헤더파일에 있는 자료형을 인식할 수가 없어서 나는 오류였던 것이었다. 그래서 include하는 순서를 바꾸어 proc.h를 가장 마지막으로 include 해주었더니 오류가 해결되었다.

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
```