Project 01 wiki

컴퓨터소프트웨어학부

2016025141

고세진

Project01 -1

명세

부모 프로세서의 아이디를 가져오는 getppid() 시스템 콜 구현하고, 이를 확인하기 위한 유저프로그램 작성

디자인

cs find t getpid

```
Cscope tag: getpid
   #   line  filename / context / line
   1      7  project01_1.c <<<unknown>>>
             printf(1,"My pid is %d\n", getpid());
   2     92  syscall.c <<<unknown>>>
             extern int sys_getpid(void);
   3    120  syscall.c <<<unknown>>>
             [SYS_getpid]  sys_getpid,
   4     12  syscall.h <<<unknown>>>
             #define SYS_getpid 11
   5     40  sysproc.c <<<unknown>>>
             sys_getpid(void)
   6     22  user.h <<<unknown>>>
             int getpid(void);
   7    434  usertests.c <<<unknown>>>
               ppid = getpid();
   8   1498  usertests.c <<<unknown>>>
                 ppid = getpid();
   9     28  usys.S <<<unknown>>>
             SYSCALL(getpid)
Type number and <Enter> (empty cancels):
```

cs find g sys_getpid 로 찾은 sys_getpid의 definition (sysproc.c)

```
sysproc.c (~/xv6-public) - VIM
37 }
38
39 int
40 sys_getpid(void)
41 {
42   return myproc()->pid;
43 }
44
```

cs find s getpid

```
Cscope tag: getpid
   #   line  filename / context / line
   1      7  project01_1.c <<main>>
             printf(1,"My pid is %d", getpid());
   2     22  user.h <<exit>>
             int getpid(void );
   3    434  usertests.c <<mem>>
             ppid = getpid();
   4   1498  usertests.c <<sbrktest>>
             ppid = getpid();
   5     28  usys.S <<SYSCALL>>
             SYSCALL(getpid)
```

cs find g proc 으로 찾은 proc.h 와 proc 구조체

```
Cscope tag: proc
   #   line  filename / context / line
   1      6  defs.h <<proc>>
             struct proc;
   2     12  proc.c <<proc>>
             struct proc proc[NPROC];
   3     10  proc.h <<proc>>
             struct proc *proc;
   4     38  proc.h <<proc>>
             struct proc {
```

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this
  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)
};
```

cscope로 pid를 얻는 getpid함수를 분석한 결과, 구조체 포인터 변수 proc 이 있다면 proc → parent → pid의 방법으로 부모 프로세스의 pid에 접근할 수 있음을 발견하였다. 따라서 다음과 같은 방법으로 project01_1을 해결할 것이다.

1. 시스템 콜의 추가

- getpid 시스템콜이 구현된 sysproc.c에 getppid 시스템콜을 구현한다.
- syscall.c, syscall.h에 getppid() 의 시스템 콜 심벌을 추가한다.
- 유저 프로그램에서 호출할 수 있도록 user.h와 usys.S에 시스템 콜과 매크로를 등록한다.

2. 유저 프로그램

- getppid()를 이용해 얻은 부모의 pid를 콘솔에 출력하여 확인할 수 있게 한다.

구현

```
93 int
94 sys_getppid(void)
95 {
96     return myproc()->parent->pid;
97 }
```

sysproc.c

```
107 extern int sys_getppid(void);
108
```

syscall.c

```
132 [SYS_getppid] sys_getppid,
133 };
```

syscall.c

```
27 int getppid(void);
```

user.h

```
24 #define SYS_getppid 23
```

syscall.h

```
33 SYSCALL(getppid)
```

usys.S

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

int
main(int argc, char* argv[]){
    printf(1,"My pid is %d\n", getpid());
    printf(1,"My ppid is %d\n", getppid());
    exit();
}
```

project01_1.c

실행 결과

getppid 시스템콜이 정상적으로 작동하여

```
$ project01_1
My pid is 11
My ppid is 2
$
```

부모 프로세스의 pid를 리턴하는 것을 확인하였음

트러블 슈팅

```
E259: no matches found for cscope query g getpid of getpid
```

- getpid()가 이름 그대로 함수로 정의되어 있는 것이 아니라서 cscope의 g 커맨드로 유의미한 결과를 얻지 못하였음
- 따라서 cs find g sys_getpid 커맨드로 분석한 결과 sys_getpid()의 형태로 정의가 되어있음을 확인하였고, 이를 기반으로 proc 구조체를 찾아나갔으며 getppid 시스템 콜을 구현할 수 있었음.

sysproc.c (~/xv6-public) - VIM

```
37 }
38
39 int
40 sys_getpid(void)
41 {
42   return myproc()->pid;
43 }
44
```

Project 01-2

명세

1. 128번 인터럽트를 발생시키는 유저 프로그램을 작성한다.
    - 유저 모드로 진입하여 커널 모드로 변경될 수 있어야 한다.
2. 128번 인터럽트의 엔트리포인트를 만든다
    - 인터럽트 발생 시 "user interrupt 128 called!"를 print 하고 프로세스가 종료된다.

디자인

1. 128번 인터럽트를 정의한다.
2. 해당 인터럽트는 유저 모드에서 엔트리 포인트에 진입할 수 있어야 한다.
    - DPL을 유저 권한으로 설정할 수 있어야 함
    - SETGATE 함수 이용
3. trap 함수에서 tf → trapno를 이용해 분기하여 핸들링한다.

- 핸들러에서는 인터럽트 번호를 프린트하고 프로세스를 종료한다.

구현

인터럽트, 핸들러 구현

1. 인터럽트 정의

```
32 #define T_MYINT         128
```

traps.h

2. SETGATE함수로 IDT와 벡터, 유저 DPL을 설정한다.

```
void
tvinit(void)
{
  int i;

  for(i = 0; i < 256; i++)
    SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
  SETGATE(idt[T_SYSCALL], 1, SEG_KCODE<<3, vectors[T_SYSCALL], DPL_USER);
  SETGATE(idt[T_MYINT], 1, SEG_KCODE<<3, vectors[T_MYINT], DPL_USER);

  initlock(&tickslock, "time");
}
```

trap.c

3. 트랩프레임에서 트랩 번호를 받아온 뒤, cprintf를 이용해 인터럽트 번호를 출력하고 exit()으로 프로세스를 종료한다.

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }
  if(tf->trapno == T_MYINT){
      cprintf("user interrupt %d called!\n", T_MYINT);
      exit();
  }
```

trap.c

유저 프로그램

```
int
main(int argc, char* argv[]){
    __asm__("int $128");
}
```

project01_2.c

실행 결과

```
$ project01_2
user interrupt 128 called!
$
```

트러블 슈팅

128번 인터럽트 핸들링 과정에서 print를 완료한 후 return을 이용해 함수를 종료하려고 하니

14번 트랩이 발생하였다. 다른 인터럽트 처리 분기에서는 exit()을 이용하여 핸들링을 종료하고 있어서 exit()을 이용하니 정상적으로 동작하였다.

cs find g exit 명령어를 이용하여 exit()의 코드를 확인해 보았을 때, 구체적인 동작 과정은 모르겠지만 프로세스의 상태와 스케줄링과 관련된 코드가 들어있었다. 따라서 이러한 처리

없이 함부로 return문을 바로 사용하게 되면 프로세스 관리에 문제가 생기므로, 정해진 방법을 이용해 프로세스를 종료해야 한다.

```
void
exit(void)
{
  struct proc *curproc = myproc();
  struct proc *p;
  int fd;

  if(curproc == initproc)
    panic("init exiting");

  // Close all open files.
  for(fd = 0; fd < NOFILE; fd++){
    if(curproc->ofile[fd]){
      fileclose(curproc->ofile[fd]);
      curproc->ofile[fd] = 0;
    }
  }

  begin_op();
  iput(curproc->cwd);
  end_op();
  curproc->cwd = 0;

  acquire(&ptable.lock);

  // Parent might be sleeping in wait().
  wakeup1(curproc->parent);

  // Pass abandoned children to init.
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->parent == curproc){
      p->parent = initproc;
      if(p->state == ZOMBIE)
        wakeup1(initproc);
    }
  }

  // Jump into the scheduler, never to return.
  curproc->state = ZOMBIE;
  sched();
  panic("zombie exit");
}

// Wait for a child process to exit and return its pid.
```

proc.c

```
$ project01_2
user interrupt 128 called!
pid 3 project01_2: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill proc
$
```

```
 if(tf->trapno == T_MYINT){
     cprintf("user interrupt %d called!\n", T_MYINT);
   return;
 }
```

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }
 if(tf->trapno == T_MYINT){
     cprintf("user interrupt %d called!\n", T_MYINT);
     exit();
 }
```

trap.c