운영체제 Project01 Wiki

2019019016 서시언

<디자인 및 구현>

[project01_1]

1. getpid()를 분석하기 위해 vim command mode에서 `:cs find t getpid` 를 입력했을 때, getpid()의 정의는 없고 wrapper함수인 sys_getpid()만 있으므로 sys_getpid()를 찾아보니, myproc()에서 리턴되는 구조체의 멤버인 pid를 리턴하는 함수임을 알 수 있다.

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

2. myproc()을 분석하기 위해 vim command mode에서 `:cs find g myproc` 을 입력해 myproc()이 정의되어 있는 부분을 찾는다.

```
// Disable interrupts so that we are not rescheduled
// while reading proc from the cpu structure
struct proc*
myproc(void) {
  struct cpu *c;
  struct proc *p;
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
  return p;
}
```

3. struct cpu의 멤버인 proc에 대해 찾아보기 위해 vim command mode에서 `:cs find g cpu` 를 입력해 struct cpu가 정의되어 있는 부분을 찾아보니, 주석을 통해 proc은 **현재 cpu에서 실행중인 프로세스의 정보**를 담고 있는 구조체의 포인터라는 것을 알 수 있다.

```
su@su: ~/os_practice/xv6-public
// Per-CPU state
struct cpu {
  uchar apicid;                // Local APIC ID
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
  int ncli;                    // Depth of pushcli nesting.
  int intena;                  // Were interrupts enabled before pushcli?
  struct proc *proc;           // The process running on this cpu or null
};
```

4. struct proc을 분석하고, pid의 의미를 알기 위해 vim command mode에서 `:cs find g proc` 을 입력해 proc이 정의되어 있는 부분을 찾는다.

```
// Per-process state
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)
};
```

➔ 주석을 통해 pid는 process ID라는 것을 알 수 있고, parent process의 정보를 저장하는 parent라는 멤버를 가지므로 지금까지 얻은 정보를 활용해 시스템콜인 fork(), exit(), wait()

가 정의되어 있는 proc.c안에 getppid()를 정의한다.

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

➔ getppid()는 myproc()을 통해 현재 실행 중인 프로세서의 정보를 담고있는 구조체의 포인터를 얻고, 이것의 멤버인 parent를 통해 부모 프로세서의 정보를 담고 있는 구조체의 포인터를 얻은 뒤, 그것의 pid를 리턴한다.

5. 다른 커널 영역의 c file에서 getppid()를 찾을 수 있게 하기 위해 defs.h에 `int getppid(void);` 를 추가해준다.

```
//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn))
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             getppid(void);
```

6. 시스템 콜의 wrapper function들이 모여있는 sysproc.c에 getppid의 wrapper함수를 만들어준다.

```
int
sys_getppid(void)
{
        return getppid();
}
~
```

7. getppid()를 system call로 등록하기 위해 syscall.h에 `#define SYS_getppid 23` 를 추가하고, syscall.c에 `extern int sys_getppid(void);` 와 `[SYS_getppid] sys_getppid,` 를 추가한다.

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_myfunction 22
#define SYS_getppid 23
```

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_myfunction(void);
extern int sys_getppid(void);
```

```
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_myfunction]      sys_myfunction,
[SYS_getppid]   sys_getppid,
};
```

8. 등록한 system call을 user program에서 사용할 수 있도록 user.h에 `int getppid(void);` 를 추가해준다.

```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int myfunction(char*);
int getppid(void);
```

9. getppid를 사용했을 때의 preprocess과정을 define하기 위해 usys.S에 `SYSCALL(getppid)` 를 추가해준다.

```
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(myfunction)
SYSCALL(getppid)
```

10. getppid()가 잘 작동하는지 확인하기 위해 project01_1.c (유저프로그램)를 작성한다.

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

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

➔ getpid()를 이용해 현재 프로세서의 id를 가져오고, getppid()를 이용해 부모 프로세서의 id를 가져와 출력한 뒤, 프로세스를 종료한다.

11. Makefile에 다음과 같이 `_project01_1\` 과 `project01_1.c` 를 추가한다.

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _my_userapp\
        _project01_1\
```

```
EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
        printf.c umalloc.c my_userapp.c project01_1.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\
```

[project01_2]

1. trap.c의 tvinit()에서 switching entrypoint from user to kernel인 T_SYSCALL(64)은 따로 빼서 초기화하는 것을 알 수 있다. 따라서 128도 따로 빼서 마찬가지로 초기화해준다.

```
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[128],1, SEG_KCODE<<3, vectors[128], DPL_USER);
  initlock(&tickslock, "time");
}
```

2. trap.c의 trap()에서 interrupt 128을 처리하기 위해 tf->trapno가 128인 경우, “user interrupt (tf->trapno) called!”를 프린트하고 exit() system call을 호출해 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 == 128){
    cprintf("user interrupt %d called!\n",tf->trapno);
    exit();
    return;
  }
```

3. “int 128”을 invoke하는 유저프로그램을 작성한다. (project01_2.c)

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

int
main(int argc, char *argv[])
{
        __asm__("int $128");
        return 0;
}
```

4. Makefile에 다음과 같이 `_project01_2\` 과 `project01_2.c` 를 추가한다.

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _my_userapp\
        _project01_1\
        _project01_2\
```

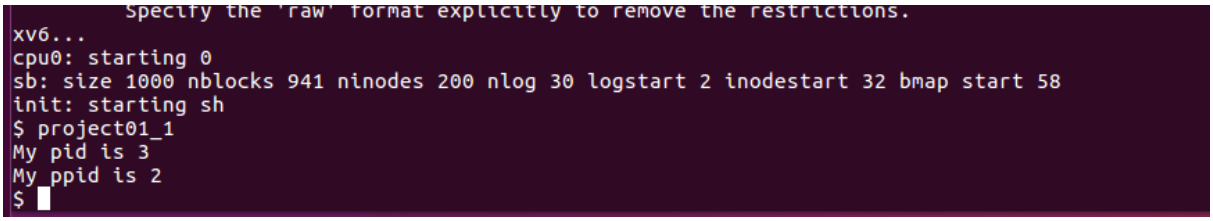
```
EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
        printf.c umalloc.c my_userapp.c project01_1.c project02_2.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\
```

<실행 결과>

[project01_1]

아래 명령어들을 순서대로 입력하면 사진과 같은 결과가 나온다.

1. make clean
2. make
3. make fs.img
4. ./bootxv6.sh
5. project01_1

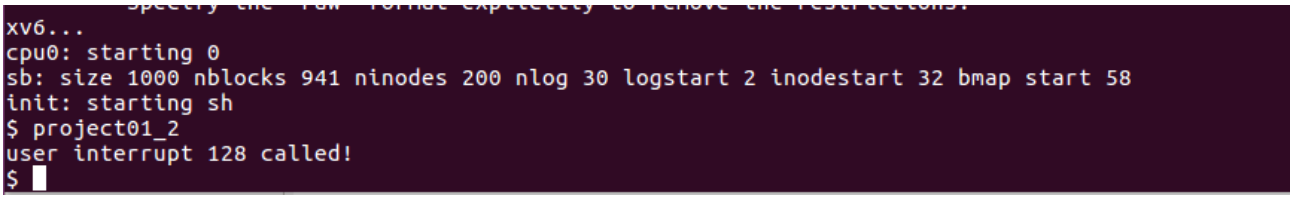


➔ project01_1.c의 main함수가 실행되어 getpid()로 현재 프로세서의 id, 3을, getppid()로 부모 프로세서의 id인 2를 받아 출력한다.

[project01_2]

아래 명령어들을 순서대로 입력하면 사진과 같은 결과가 나온다.

1. make clean
2. make
3. make fs.img
4. ./bootxv6.sh
5. project01_2



➔ interrupt 128이 invoke되어 trap()에서 “user interrupt (trap number) called!”를 프린트하고, exit()를 호출해 프로세스를 종료한다.

<트러블슈팅>

-project01_1.c를 왼쪽과 같이 작성한 뒤 실행하면 오른쪽과 같은 에러가 발생했다.

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

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

```
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ project01_1
My pid is 3
My ppid is 2
pid 3 project01_1: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill p
roc
$
```

➔ 다음과 같이 exit()를 추가해 프로세스를 종료해주자 에러가 발생하지 않았다.

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

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

-trap.c의 trap()에서 trapno가 128인 경우 다음과 같이 처리했음에도 “user interrupt (trap number) called!”가 프린트되지 않고 오른쪽과 같은 에러가 발생했다.

```c
  if(tf->trapno == 128){
    cprintf("user interrupt %d called!\n",tf->trapno);
    exit();
    return;
  }
```

```
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ project01_2
pid 3 project01_2: trap 13 err 1026 on cpu 0 eip 0x3 addr 0x0--kill proc
$
```

➔ trap.c의 tvinit()에서 switching entrypoint from user to kernel인 T_SYSCALL과 마찬가지로 interrupt 128을 따로 빼서 처리를 해주자 에러가 발생하지 않았다.

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

  for(i = 0; i < 256; i++)
    SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
  SETGATE(idt[128],1, SEG_KCODE<<3, vectors[128], DPL_USER);
  initlock(&tickslock, "time");
}
```