운영체제 Assignment 1 Wiki

이름: 권도현

학번: 2023065350

학과: 컴퓨터소프트웨어학부

Design

- **Assignment goal**

  - 이 과제의 목적은 User level에서 "ppid" 명령어를 실행하면, 시스템 내부적으로 kernel level의 getppid() system call을 호출하여, Parent process id (ppid)를 반환하는 것이다.

- **Plan**

  1. getppid()의 경우, 이미 구현되어져 있는 getpid() system call과 유사하다.

  2. kernel/sysproc.c 안에 있는 getpid()의 구조가 아래와 같고, myproc()에 대한 정보가 필요함을 확인한다.

```
uint64
sys_getpid(void)
{
  return myproc()->pid;
}
```

  3. kernel/proc.c에서 myproc()를 확인한다.

```
// Return the current struct proc *, or zero if none.
struct proc*
myproc(void)
{
  push_off();
  struct cpu *c = mycpu();
  struct proc *p = c->proc;
  pop_off();
  return p;
}
```

  myproc() 함수는 현재 cpu의 process를 꺼내어 반환해준다.

4. kernel/proc.h에서 proc struct의 구조를 확인한다.

```
// Per-process state
struct proc {
  struct spinlock lock;

  // p->lock must be held when using these:
  enum procstate state;        // Process state
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  int xstate;                  // Exit status to be returned to
parent's wait
  int pid;                     // Process ID

  // wait_lock must be held when using this:
  struct proc *parent;         // Parent process

  // these are private to the process, so p->lock need not be he
ld.
  uint64 kstack;               // Virtual address of kernel stac
k
  uint64 sz;                   // Size of process memory (bytes)
  pagetable_t pagetable;       // User page table
  struct trapframe *trapframe; // data page for trampoline.S
  struct context context;      // swtch() here to run process
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Proc 구조체 내에 proc* parent로 parent process를 가리키고 있다.

5. 직접 getppid() 구현하기에 앞서, user level에서 "ppid"라는 명령어를 인식하여 getppid()를 호출하도록 구현하기 위해 아래와 같은 행동을 해야 된다.

    a. User level에서 "ls", "rm" 과 같은 명령어가 인식되기 위해 ppid.c 파일을 만들어야 한다.

    b. getpid()와 같이 getppid()가 system call으로써 동작할 수 있게 syscall.c / syscall.h에 getppid()를 추가해야 하고, sysproc.c에 getppid() 함수를 구현해야 한다.

Implementation

1. kernel/syscall.c

```
extern uint64 sys_getppid(void); // Newly added
```

다른 파일에서 정의된 sys_getppid() 함수를 참조한다.

```
[SYS_getppid] sys_getppid, // Newly added
```

syscall.h에서 정의할 system call number와 매칭시키기 위한 코드

2. kernel/syscall.h

```
#define SYS_getppid 22 // Newly added
```

System call number를 지정

3. kernel/sysproc.c

```
uint64 // Newly added
sys_getppid(void)
{
  return myproc() -> parent -> pid;
}
```

앞서 Design 단계에서 확인한 것과 같이 myproc()는 현재 proc struct를 가리키는 포인터를 반환하고, proc struct는 *parent와 pid를 struct member로 가지고 있음으로 위와 같이 구현 가능하다.

4. user/user.h

```
int getppid(void); // Newly added
```

getppid()를 user program에 등록한다.

5. user/usys.pl

```
entry("getppid");
```

usys.S에 getppid 에 대한 system call interface를 만드는 어셈블리 코드를 만들도록 해준다. usys.S 에 추가된 코드는 아래와 같다.

```
.global getppid
getppid:
 li a7, SYS_getppid
 ecall
 ret
```

6. user 폴더에 ppid.c 파일 생성

```c
#include "kernel/types.h"
#include "user/user.h"

int main() {
        int pid = getpid();
        int ppid = getppid();
        printf("My student ID is 2023065350\n");
        printf("My pid is %d\n", pid);
        printf("My ppid is %d\n", ppid);
}
```

xv6 shell에서 ppid 라는 명령어를 사용할 수 있도록 하기 위해서 필요하다.

7. Makefile/UPROGS에 ppid 추가

```
UPROGS=\
        $U/_cat\
        $U/_echo\
        $U/_forktest\
        $U/_grep\
        $U/_init\
        $U/_kill\
        $U/_ln\
        $U/_ls\
        $U/_mkdir\
        $U/_rm\
        $U/_sh\
        $U/_stressfs\
        $U/_usertests\
        $U/_grind\
        $U/_wc\
        $U/_zombie\
        $U/_ppid\
```

Makefile에 user program에 대한 정보를 추가한다.

Results

1. **make clean**

```
kwond@NOTKDH:~/assignment-2023065350$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*/*.o */*.d */*.asm */*.sym \
user/initcode user/initcode.out kernel/kernel fs.img \
mkfs/mkfs .gdbinit \
        user/usys.S \
user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh use
r/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_ppid
```

2. **make qemu**

```
kwond@NOTKDH:~/assignment-2023065350$ make qemu
riscv64-linux-gnu-gcc    -c -o kernel/entry.o kernel/entry.S
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -fno-common -nostdlib
 -fno-builtin-strncpy -fno-builtin-strncmp -fno-builtin-strlen -fno-builtin-memset -fno-builtin-memmove -fno-builtin-mem
```

3. **ppid on xv6 shell: 실행 결과**

```
xv6 kernel is booting

init: starting sh
$ ppid
My student ID is 2023065350
My pid is 3
My ppid is 2
```

Troubleshooting

1. **처음에 코드를 짤 때, proc.c와 sysproc.c을 제대로 구별하지 못하여 sys_getpid() 함수를 proc.c에서 계속 찾았다.**
   A. 검색을 통해 찾아본 결과, proc.c는 kernel 내부에서 process를 관리하는 함수를 모아놓은 파일, sysproc.c는 user level에서 호출할 수 있는 system call을 구현하는 파일이였다.

   B. sysproc.c를 확인해보니 sys_getpid() 함수를 확인할 수 있었고, sys_getppid()를 구현할 수 있었다.

2. **이론 수업에서 들었던 것처럼, system call number를 지정해줘야 하는 이유와 코드 내에서 지정하는 위치는 알지만 kernel 내부에서 system call number의 해석을 어디가 담당하는지 찾을 수 없었다.**
   A. kernel/syscall.c 내부의 syscall() 함수에서 해당 기능을 수행하고 있었다.

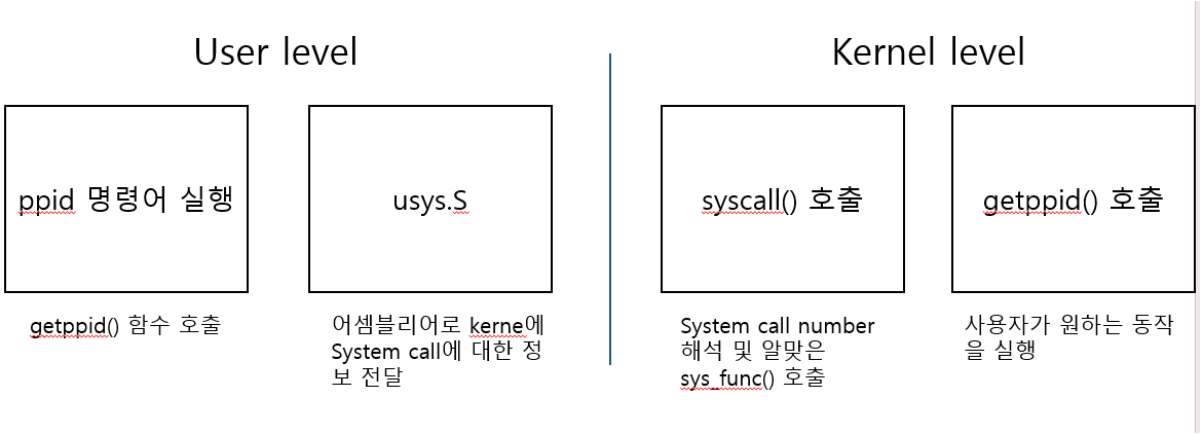
```
void
syscall(void)
{
  int num;
  struct proc *p = myproc();

  num = p->trapframe->a7;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    // Use num to lookup the system call function for num, call
it,
    // and store its return value in p->trapframe->a0
    p->trapframe->a0 = syscalls[num]();
  } else {
    printf("%d %s: unknown sys call %d\n",
            p->pid, p->name, num);
    p->trapframe->a0 = -1;
  }
}
```

   B. num = p->trapframe->a7 이 user level에서 a7 레지스터에 저장한 syscall number를 꺼내 오는 코드인 것을 확인할 수 있었지만 이미 다 구현된 부분이라 과제 해결과는 연관이 없었다.

3. **Makefile을 수정하는 과정에서 슬래쉬(/) 와 역슬래쉬(₩)를 잘못 사용해서 아래와 같은 오류가 발생했다.**
   A. 단순 오타로 인한 오류라 오타를 고쳐 해결했다.

```
make: *** No rule to make target 'user/_ppid/', needed by 'fs.im
g'.  Stop.
```

Attachment: getppid() system call process



위의 동작을 기반으로, 내가 과제를 수행하며 수정한 코드들이 왜 그렇게 구현되거나 추가되어야 하는지를 살펴보자.

1. **syscall.h**
   A. syscall()에서 system call number를 해석할 때 필요한 정보를 정의한다.

2. **syscall.c**
   A. system call number와 sys_function을 mapping 해준다.

3. **sysproc.c**
   A. 사용자가 실제로 원하는 동작을 구현한다.

4. **user.h**
   A. user level에서 호출할 함수를 정의한다.

5. **usys.pl**
   A. usys.S 파일에 macro가 생성되도록 한다.

6. **ppid.c**
   A. ppid 명령어에 대해 실제 실행될 파일을 추가한다.

7. **Makefile**
   A. ppid 명령어가 쉘에서 실행될 수 있도록 명령어를 등록 및 컴파일한다.

Question

User level에서 사용하는 getppid() 함수와, Kernel level에서 사용하는 sys_getppid() 함수는 이름이 다른데 어떻게 연속하여 실행되는걸까?

1. getppid() 함수는 실제로 사용자가 원하는 역할을 수행해주지 못하고 레지스터에 정보를 추가하여 kernel에 넘겨주고 trap을 발생시켜 kernel mode로 들어가는 껍데기 함수이다.

2. usys.S에서 자동으로 구현된 getppid() 함수는 (1)의 작업을 수행한다.

3. Kernel 내부에서 구현된 mapping table로 인하여 자동으로 알맞은 system call function을 호출할 수 있다.
   A. 즉, user level의 function은 중요치 않다.

소감: getpid() 함수를 참고하여 과제 해결을 하며 궁금했던 부분을 해결할 수 있었다.