CS 238P
Operating Systems
Discussion 9

Today’s agenda

- Creating time system call

What is system call

- Call of a kernel level function
- Done by interrupts or *sysenter* (newer hardware)
- Linux uses *int 0x80*, xv6 uses *int 0x40*
- Stack is separate from user program
- Way more expensive than a normal call

What is system call

- Each syscall is associated with some number
- If you call a syscall from userspace the call looks like that (syscall MY_SYSCALL with number 0x1):

```
// saving registers on stack

…

  .globl MY_SYSCALL; \

  MY_SYSCALL: \

    movl 0x1, %eax; \

    int 0x40; \

    ret
```

Implementing new syscall

1. Add new system call number in *syscall.h*
2. Declare your syscall using *extern int sys_CALLNAME(void);* in *syscall.c*
3. Link syscall number with function in *syscalls.c* array *syscalls*
4. Register your call in userspace in *user.h*
5. Register syscall in *usys.c*
6. Implement your system call in one of .c files (for example *sysproc.c*)

How to get arguments

Get integer:

```
int argint(int n, int *ip)
```

n is argument position

ip is location where to store argument

Example (get first argument of syscall and store it in pid variable):

```
  int pid;
  if(argint(0, &pid) < 0)
    return -1;
```

How to get arguments

Get pointer:

```
int argptr(int n, char **pp, int size)
```

n is argument position

pp is location where to store argument

size is size of the array in bytes

Example (get second argument of syscall and store it in arr variable):

```
  struct stat *st;
  if(argptr(1, (void*)&st, sizeof(*st)) < 0)
    return -1;
```

How to get arguments

Get string:

```
int argstr(int n, char **pp)
```

n is argument position

pp is location where to store argument

Example (get second argument of syscall and store it in str variable):

```
  char *old;
  if(argstr(1, &old) < 0)
    return -1;
```

How to return data back?

- Return code: just return int from syscall
- For more complex data - store them in the passed argument

Cool, I implemented my syscall but how to test it?

- Create a user program which calls it
  - Create a file with your program (for example *mytestprogram.c*)
  - Add your program into *UPROGS* in *Makefile*
  - Add your program into *EXTRA* in *Makefile*
  - Rebuild Qemu