xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

System calls

Carmi Merimovich

Tel-Aviv Academic College

November 30, 2017

syscall()

- **syscall()** dispatches to the function implementing the relevant system call.
- System call number in user mode is in **eax**.
- Hence it is in **myproc()**->**tf**->**eax**.
- We will be done by having **syscalls[]**, a vector of routines addresses.
- return value of **syscalls[myproc()**->**tf**->**eax]** is put into **myproc()**->**tf**->**eax**.

syscall.h

```
#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
```

syscalls[]

```
3350 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,
};
```

syscall()

```
3701 void syscall(void) {
      int num;

      num = myproc()->tf->eax;
      if (num > 0 && num < NELEM(syscalls) &&
                syscalls[num]) {
        myproc()->tf->eax = syscalls[num]();
      } else {
        cprintf("%d %s: unknown sys call %d\n",
        myproc()->pid, myproc()->name, num);
        myproc()->tf->eax = -1;
      }
    }
```

myproc()->tf ->eax = syscalls[num]()

```
switch (num) {
case SYS_fork:
  myproc()->tf->eax = sys_fork();
  break;
case SYS_exit:
  myproc()->tf->eax = sys_exit();
  break;
case SYS_mkdir:
  myproc()->tf->eax = sys_mkdir()
  break;
case SYS_close:
  myproc()->tf->eax = sys_close();
  break;
default:
 myproc()->tf->eax = -1;
 break
}
```

We can investigate each system call now...

Process control syscalls

Our current knowledge is enough for the first two syscalls:

- **getpid()**.
- **fork()**.

In addition, for the following syscalls we need to access arguments:

- **kill()**.

In addition, for the following we also need the event system:

- **exit()**.
- **wait()**.
- **sleep()**.

In addition, for the following syscall we need to read file:

- **exec()**.