Operating System

Project #1-1

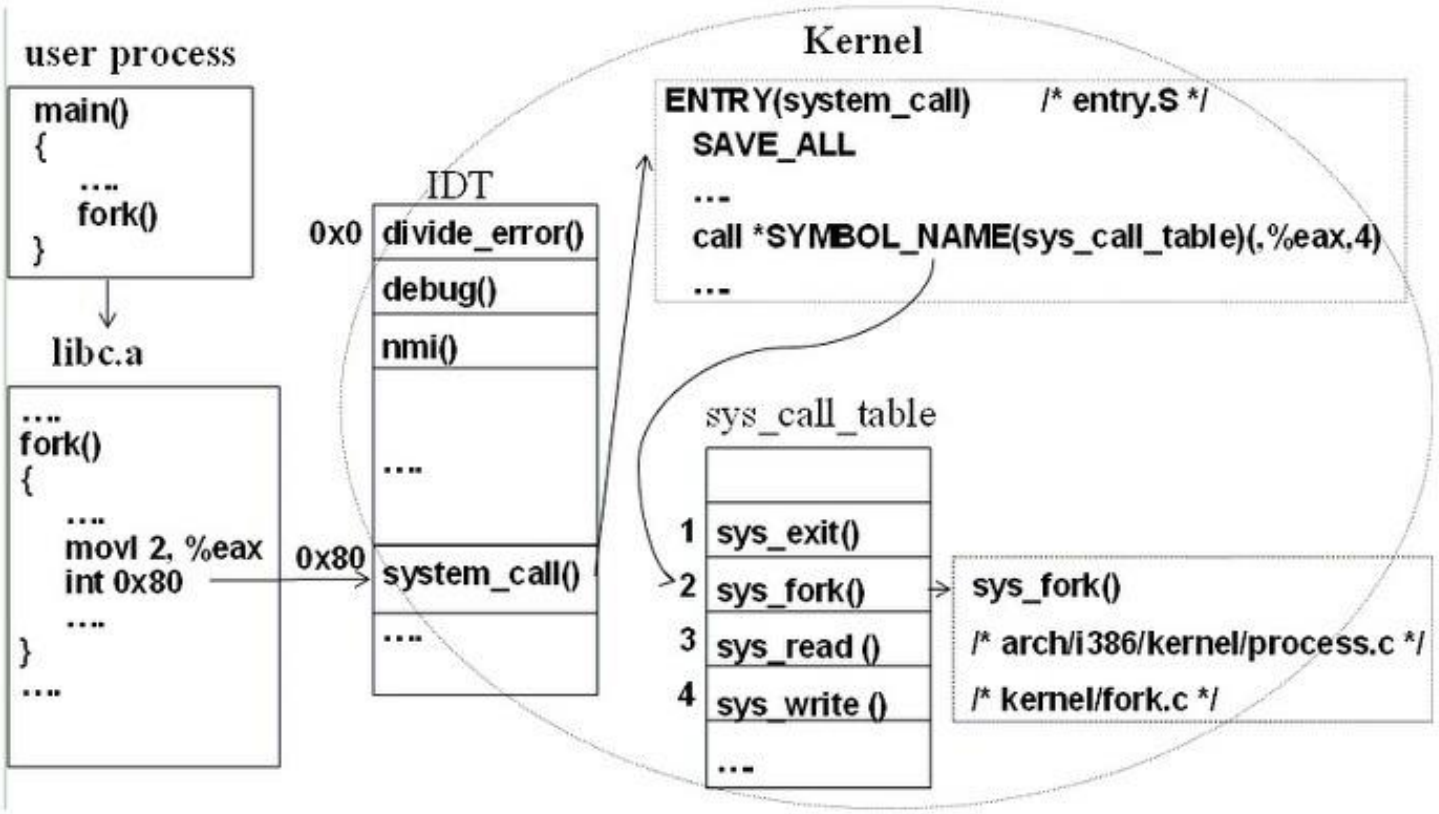
16.09.19

Project Plan

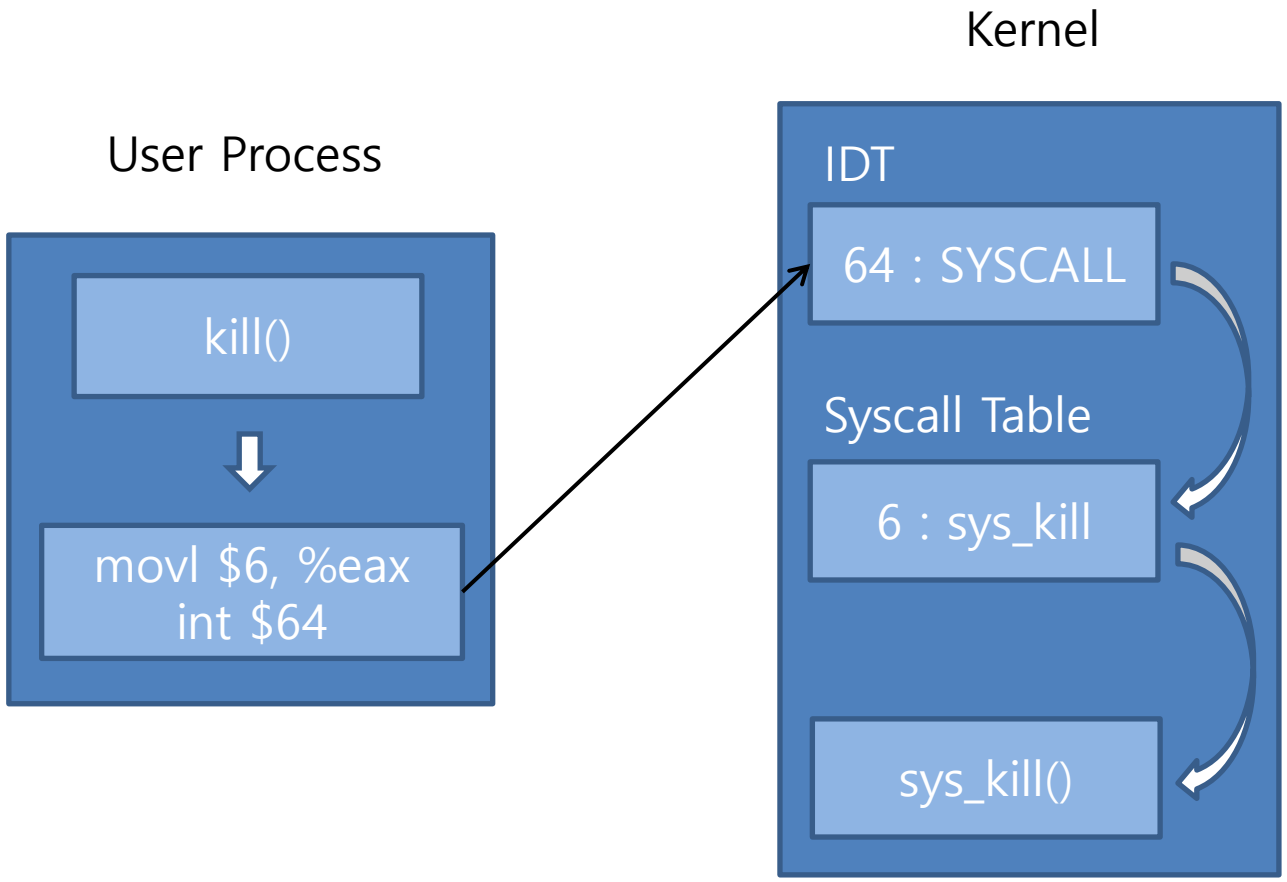
- 5 projects
  - Install Xv6
  - System call + scheduling
  - Thread-support
  - Virtual memory
  - Concurrency

- Single-handed project

System Call Process (Linux)



System Call Process on Xv6



System Call Process on Xv6 (cont.)

- user.h

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

System Call Process on Xv6 (cont.)

- defs.h

```
// proc.c
struct proc*     copyproc(struct proc*);
void             exit(void);
int              fork(void);
int              growproc(int);
int              kill(int);
```

System Call Process on Xv6 (cont.)

- usys.S

```
#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret
```

```
SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getppid)
```

System Call Process on Xv6 (cont.)

- syscall.h

```
// 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_getppid 22
```

System Call Process on Xv6 (cont.)

- traps.h

```
// x86 trap and interrupt constants.

// Processor-defined:
#define T_DIVIDE         0      // divide error
#define T_DEBUG          1      // debug exception
#define T_NMI            2      // non-maskable interrupt
#define T_BRKPT          3      // breakpoint
#define T_OFLOW          4      // overflow
#define T_BOUND          5      // bounds check
#define T_ILLOP          6      // illegal opcode
#define T_DEVICE         7      // device not available
#define T_DBLFLT         8      // double fault
// #define T_COPROC      9      // reserved (not used since 486)
#define T_TSS           10      // invalid task switch segment
#define T_SEGNP         11      // segment not present
#define T_STACK         12      // stack exception
#define T_GPFLT         13      // general protection fault
#define T_PGFLT         14      // page fault
// #define T_RES        15      // reserved
#define T_FPERR         16      // floating point error
#define T_ALIGN         17      // aligment check
#define T_MCHK          18      // machine check
#define T_SIMDERR       19      // SIMD floating point error

// These are arbitrarily chosen, but with care not to overlap
// processor defined exceptions or interrupt vectors.
#define T_SYSCALL       64      // system call
#define T_DEFAULT      500      // catchall
```

System Call Process on Xv6 (cont.)

- trap.c

```
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);

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

System Call Process on Xv6 (cont.)

- vectors.S

```
# vector table
.data
.globl vectors
vectors:
  .long vector0
  .long vector1
  .long vector2
  .long vector3
  .long vector4
  .long vector5
  .long vector6
  .long vector7
  .long vector8
```

```
vector64:
  pushl $0
  pushl $64
  jmp alltraps
```

System Call Process on Xv6 (cont.)

- trapasm.S

```
  # vectors.S sends all traps here.
.globl alltraps
alltraps:
  # Build trap frame.
  pushl %ds
  pushl %es
  pushl %fs
  pushl %gs
  pushal

  # Set up data and per-cpu segments.
  movw $(SEG_KDATA<<3), %ax
  movw %ax, %ds
  movw %ax, %es
  movw $(SEG_KCPU<<3), %ax
  movw %ax, %fs
  movw %ax, %gs

  # Call trap(tf), where tf=%esp
  pushl %esp
  call trap
  addl $4, %esp
```

System Call Process on Xv6 (cont.)

- trap.c

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(proc->killed)
      exit();
    proc->tf = tf;
    syscall();
    if(proc->killed)
      exit();
    return;
  }
```

System Call Process on Xv6 (cont.)

- syscall.c

```
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,
```

```
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
```

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

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

System Call Process on Xv6 (cont.)

- sysproc.c

```
int
sys_kill(void)
{
  int pid;

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

- proc.c

```
int
kill(int pid)
{
  struct proc *p;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      p->killed = 1;
      // Wake process from sleep if necessary.
      if(p->state == SLEEPING)
        p->state = RUNNABLE;
      release(&ptable.lock);
      return 0;
    }
  }
  release(&ptable.lock);
  return -1;
}
```

Project #1-1 – System Call

- Make two system calls (getnice, setnice)
- Synopsis
  - int getnice(int pid);
  - int setnice(int pid, int value);
- Description
  - getnice() : Obtain the nice value of process
  - setnice() : set the nice value of process
  - The default nice value is 20
  - The range of valid nice values is [0,40]
- Return Value
  - getnice() : Return the nice value of process on success. Return -1 if there is no process corresponding to the pid
  - setnice() : Return 0 on success. Return -1 if there is no corresponding to the pid or the nice value is out of range

Project #1-1 – System Call (cont.)

- Checkpoint
  - Get nice value of init process (must 20) – **10p**
  - Get nice value of non-existing process (wrong pid) – **20p**
  - Set nice value of current process – **20p**
  - Set nice value of non-existing process (wrong pid) – **20p**
  - Set wrong nice value of current process (out of nice range) – **20p**
  - Get nice value of forked process – **10p**

Project #1-1 – System Call (cont.)

- Submit a tar.gz file

- Send email to T.A
  - [SSE3044]Project#1_1-YOURID-YOURNAME.tar.gz
    - ex) [SSE3044]Project#1_1-2016710580-이규선.tar.gz
  - Email address : lgs0409@naver.com
  - Wrong title is not allowed

- Due date
  - 2016-09-25(Sun) PM 23:59
  - Get no point for late submission

Add Testcase

- Makefile

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

- test.c

```c
#include "syscall.h"
#include "traps.h"
#include "types.h"
#include "user.h"

int main(int argc, char **argv)
{
    exit();
}
```

Add Testcase (cont)

- make → make qemu