ADDING SYSTEM CALLS IN XV6.

Dr. Mayank Pandey
Motilal Nehru National Institute of Technology Allahabad, Prayagraj

- A system call is simply a kernel function that a user application can use to access or utilize system resources.
- Functions fork(), and exec() are well-known examples of system calls in UNIX and xv6.
- An application signals the kernel it needs a service by issueing a software interrupt, a signal generated to notify the processor that it needs to stop its current task, and response to the signal request.

- Before switching to handling the new task, the processor has to save the current state, so that it can resume the execution in this context after the request has been handled.
- The following is a code that calls a system call in xv6 (found in initcode.S):

```
.globl start
start:
  pushl $argv
  pushl $init
  pushl $0  // where caller pc would be
  movl $SYS_exec, %eax
  int $T_SYSCALL
```

- Basically, it pushes the argument of the call to the stack, and puts the system call number, which is $SYS_exec in the example, into %eax.
- All the system call numbers are specified and saved in a table and the system calls of xv6 can be found the file syscall.h
- Next, the code int $T_SYSCALL generates a software interrupt, indexing the interrupt descriptor table to obtain the appropriate interrupt handler.

- The function trap() (in trap.c) is the specific code that finds the appropriate interrupt handler.
- It checks whether the trap number in the generated trapframe
- Trapframe : (a structure representing the processor's state at the time the trap happened) is equal to T_SYSCALL.

- If it is, it calls syscall(), the software interrupt handler that's available in **syscall.c**.

```
// This is the part trap that calls syscall()
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(proc->killed)
      exit();
    proc->tf = tf;
    syscall();
    if(proc->killed)
      exit();
    return;
  }
  .....
}
```

- The function syscall() is the final function that checks out %eax to obtain the system call's number, which is used to index the table with the system call pointers, and to execute the code corresponding to that system call:

```
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;
  }
}
```

- The following are the procedures of adding our exemplary system call cps() to xv6.
- Add name to **syscall.h:**

```
// System call numbers
#define SYS_fork    1
.........
#define SYS_close  21
#define SYS_cps    22
```

- Add function prototype to defs.h:

```
// proc.c
void            exit(void);
......
void            yield(void);
int             cps ( void );
```

- Add function prototype to user.h:

```
// system calls
int fork(void);
.....
int uptime(void);
int cps ( void );
```

- Add function call to **sysproc.c**:

```
int
sys_cps ( void )
{
  return cps ();
}
```

- Add call to usys.S:

  SYSCALL(cps)

- Add call to syscall.c:

```
extern int sys_chdir(void);
.....
extern int sys_cps(void);
.....
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
.....
[SYS_close]   sys_close,
[SYS_cps]     sys_cps,
};
```

- Add code to proc.c:

```
//current process status
int cps() {
  struct proc *p;
  // Enable interrupts on this processor.
  sti();
    // Loop over process table looking for process with pid.
  acquire(&ptable.lock);
  cprintf("name \t pid \t state \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if ( p->state == SLEEPING )
        cprintf("%s \t %d  \t SLEEPING \n ", p->name, p->pid );
      else if ( p->state == RUNNING )
        cprintf("%s \t %d  \t RUNNING \n ", p->name, p->pid );
  }
  release(&ptable.lock);
  return 22;
}
```

- Create testing file ps.c with code shown below:

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
int
main(int argc, char *argv[])
{
  cps();

  exit();
}
```

- Modify Makefile : Modify Makefile to include ps.c as discussed in class.

- After you have compiled and run "$make qemu-nox", you can execute the command "$ps" inside xv6. You should see outputs similar to the following:

```
 name   pid   state
init    1     SLEEPING
 sh     2     SLEEPING
 ps     3     RUNNING
```