OS Task 3: Adding System Calls In XV6

Farah Kabesh - 900191706
Karim Sherif - 900191474

Description of 3 System Calls & Prototypes

sys_cps
sys_addrsize
sys_datetime

sys_cps

- Prototype: `int sys_cps(void)`
- Purpose: gets process name, process ID and process state
- Implementation:

```c
int
cps(void)
{
        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 \t \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state == SLEEPING)
    cprintf("%s \t %d \t SLEEPING \t \n ", p->name, p->pid);
  else if(p->state == RUNNING)
    cprintf("%s \t %d \t RUNNING \t \n ", p->name, p->pid);
  else if(p->state == RUNNABLE)
    cprintf("%s \t %d \t RUNNABLE \t \n ", p->name, p->pid);
  }
  release(&ptable.lock);
  return 22;
}
```

Testing The System Call

QEMU

Machine View

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00



Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ ps
name o pid o state o
init o 1 o SLEEPING o
 sh o 2 o SLEEPING o
 ps o 3 o RUNNING o
 $
```

sys_addrsize

- Prototype: `int sys_addrsize(void)`
- Purpose: gets address space size of process memory (currently running user program) in bytes
- Implementation:

```
int
sys_addrsize(void)
{
        return myproc()->sz;
}

```

Testing The System Call

QEMU

Machine View

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00



Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ addressSize
Address Space Size Is: 12288 bytes
$ _
```

sys_datetime

- Prototype: `int sys_datetime(void)`
- Purpose: gets current date and time (UTC timezone)
- Implementation:

```
int
sys_datetime(void)
{
        struct rtcdate *d;
        if(argptr(0, (void*) &d, sizeof(struct rtcdate)) < 0)
                return -1;
        cmostime(d);

        return 0;
}

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

- *cmostime()* helper function which is defined in *lapic.c* file to read real time clock

- *date.h* file contains definition of struct *rtcdate*

Testing The System Call

QEMU

Machine View

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00



Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ getDT
Date Is: 3/10/2022 And Time Is: 21:1:44$
```

Reasons Why 3 Functions Should Be System Calls

- The Kernel mode allows programs to access memory and hardware resources directly.
- Therefore, when a program needs a memory or resource, it makes a system call and the mode changes from user mode to kernel mode.
- System calls are handled via a software interrupt.
- The interrupt handler runs the implementation of the system call in kernel mode, then switches back to user mode.

How System Calls Were Added In XV6

- Edit *syscall.h* file

*syscall.h
~/xv6

```
// 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_addrsize 22
#define SYS_datetime 23
#define SYS_cps  24
```

How System Calls Were Added In XV6

- Edit *syscall.c* file to include function prototypes and add them to array of syscalls

```
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_addrsize(void);
extern int sys_datetime(void);
extern int sys_cps(void);

```

```
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,
[SYS_addrsize] sys_addrsize,
[SYS_datetime] sys_datetime,
[SYS_cps] sys_cps,
};
```

How System Calls Were Added In XV6

- Edit *sysproc.c* file to implement system call functions

```c

int
sys_addrsize(void)
{
        return myproc()->sz;
}

int
sys_datetime(void)
{
        struct rtcdate *d;
        if(argptr(0, (void*) &d, sizeof(struct rtcdate)) < 0)
                return -1;
        cmostime(d);

        return 0;
}

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

C ▾ Tab Width: 8 ▾ Ln 16, Col 4 ▾ INS

How System Calls Were Added In XV6

sys_cps

- Edit *defs.h* and *proc.c* files

```c
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             cps(void);

```

```c

int
cps(void)
{
        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 \t \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state == SLEEPING)
    cprintf("%s \t %d \t SLEEPING \t \n ", p->name, p->pid);
  else if(p->state == RUNNING)
    cprintf("%s \t %d \t RUNNING \t \n ", p->name, p->pid);
  else if(p->state == RUNNABLE)
    cprintf("%s \t %d \t RUNNABLE \t \n ", p->name, p->pid);
  }
  release(&ptable.lock);
  return 22;
}
```

How System Calls Were Added In XV6

- Edit *usys.s* and *user.h* files

```
#include "syscall.h"
#include "traps.h"

#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 (addrsize)
SYSCALL (datetime)
SYSCALL (cps)
```

```
struct stat;
struct rtcdate;

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

Description of 3 User Programs

- User Program Name: **ps.c**
- Calls system call *cps()* to test its functionality

```c
#include "types.h"
#include "user.h"
#include "stat.h"



int main()
{
        cps();
        exit();
}

```

Description of 3 User Programs

```c
#include "types.h"
#include "user.h"
#include "stat.h"

int main(void)
{
        printf(1, "Address Space Size Is: %d bytes \n", addrsize());

        exit();
}

```

- User Program Name: **addressSize.c**
- Calls system call *addrsize()* to test its functionality and prints address space size in bytes

Description of 3 User Programs

- User Program Name: **getDT.c**
- System call *datetime(&s)* used inside an if-statement to test its functionality. If equal to -1, then it'll print "Failed To Obtain Date/Time". Otherwise it'll print Date and Time in the specified format.
- (s.hour + 2) in order to display time in Egypt, since it is automatically set to the UTC timezone .

Open *getDT.c ~/xv6

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "date.h"

int main (int argc, char *argv[])
{
        struct rtcdate s;

        if(datetime(&s)==-1)
        {
                printf(2, "Failed To Obtain Date/Time");
                exit();
        }

        printf(1, "Date Is: %d/%d/%d And Time Is: %d:%d:%d", s.day, s.month, s.year, (s.hour + 2), s.minute, s.second);
        exit();
}
```