Lab #1: System Call Implementation

Hongbo Qin X674211
Shuai Wang X674711

17 April 2020
Dr. Hyoseung Kim

CS202 Advanced Operating System

1. The List of the Modified Files

proc.c, test.c, Makefile, defs.h, proc.h, syscall.c, syscall.h, sysproc.c, user.h, usys.S,

2. Ideas Illustration

[try to explain the idea in high level]

2.1 Count the number of processes in the system

The first task for the info() system call is getting the total number of processes in the xv6 system. There is a process table called ptable in xv6. Each process has its own current state, including *UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE.* What we do is counting all the processes whose state is not UNUSED or ZOMBIE.

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

proc.h [35]

2.2 Count the total number of system calls that the current process has made

In this task we create a new variable called “syscallCounter” for each process to track the total number of system calls it made. We can update this variable in syscall() function.

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();
  //------cs202-------//
  curproc->syscallCounter++;

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

syscall.c [134-150]

2.3 Count the number of presenting memory pages

The task asks us to find the total number of the presenting memory pages for the current process. According to the official documentation [Chapter 3.6] , the system will automatically set the PTE_W, PTE_X, PTE_R, PTE_U, and PTE_V flags in the page table entries. While we are only concerned with the PTE_P flag, which indicates whether the page table entry is present in the memory or not.

```
// Page table/directory entry flags.
#define PTE_P           0x001   // Present
#define PTE_W           0x002   // Writeable
#define PTE_U           0x004   // User
#define PTE_PS          0x080   // Page Size
```

mmu.h [93-97]

Thus, the overall idea is that, loop through the entire page table of the process, count the total number of the pages with PTE_P set to 1.

3. Source Code Illustration

[try to explain the detailed implementation]

3.1 Create info() system call

1. In defs.h:

```
117  void            setproc(struct proc*);
118  void            sleep(void*, struct spinlock*);
119  void            userinit(void);
120  int             wait(void);
121  void            wakeup(void*);
122  void            yield(void);
123  int             info(int);
124
```

2. In syscall.c:

```
104  extern int sys_write(void);
105  extern int sys_uptime(void);
106  //-------cs202---------//
107  extern int sys_info(void);
108
```

```
128  [SYS_link]    sys_link,
129  [SYS_mkdir]   sys_mkdir,
130  [SYS_close]   sys_close,
131  [SYS_info]    sys_info,
132  };
133
```

3. In syscall.h:

```
18  #define SYS_mknod  17
19  #define SYS_unlink 18
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_info   22
```

4. In user.h:

```
22  int getpid(void);
23  char* sbrk(int);
24  int sleep(int);
25  int uptime(void);
26  int info(int);
27
```

5. In usys.S:

```
28  SYSCALL(getpid)
29  SYSCALL(sbrk)
30  SYSCALL(sleep)
31  SYSCALL(uptime)
32  SYSCALL(info)
```

6. In sysproc.c:
   We use argint() to get the argument from info() function and then call the info(param) function in proc.c.

```
10  //-----cs202------//
11  int
12  sys_info(void)
13  {
14    int param;
15    if(argint(0,&param) < 0)
16      return -1;
17    return info(param);
18  }
```

3.2 Count the number of processes in the system

We can iterate through the process table and when the state of the current iteration's process is not UNUSED or ZOMBIE, our number counter plus 1.

We need to lock the process table in order to prevent race conditions by calling acquire() function.

```
case 1:
  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == UNUSED || p->state == ZOMBIE)
      continue;
    numCounter++;
  }
  release(&ptable.lock);
 // cprintf("We have %d processes in this system\n", numCounter);
  return numCounter;
  break;
```

3.3 Count the total number of system calls that the current process has made

We have already illustrated the main idea of this part in 2.2. So in info() function, what we need to do is return the value of the current process's syscallCounter.

```
case 2:
 // cprintf("This process has made %d system calls\n",curproc->syscallCounter);
  return curproc->syscallCounter;
  break;
```

3.4 Count the number of presenting memory pages

We can loop through the page table by one simple for-clause. If the PTE_P bit is set, we will need to increment the counter by 1, which means that the current page is in the memory.

At the end, the function will directly return the counter as the number of all presenting memory pages of the current process.

```
case 3:
  // loop through the page table of the current process
  for (i = 0; i < curproc->sz; i += PGSIZE) {
    // increment the counter if the present bit is set
    if ((*walkpgdir(curproc->pgdir, (void *)i, 0)) & PTE_P) {
      numCounter++;
    }
  }
  return numCounter;
  break;
```

4. Test

We create a test file called test.c. It calls info three times with 1,2,3 as the argument.

```c
int main(int argc, char *argv[])
{
  int numberOfProcess = info(1);
  int pages = info(3);
  int syscalls = info(2);
```

After finishing all the system calls, numberOfProcess should be the number of the processes in this system. Pages is the number of the memory pages that the current process is using. Syscalls records the number of system calls made by the current process.

Result:

```
$ test
Now we have 3 processes in this system.
This process has made 3 times system calls in main() function.
The total number of the memory pages the current process is using: 3
```