

Lab 1 - System Call Implementation

Due - 04/25/23

Anthony Mendez | 862307065 | amend003

Jordan Kushner | 862294132 | jkusc002

Video Demo Link - <https://youtu.be/DVp-2Bas94s>

All Modified Files

- ~/kernel/ files:
 - Defs.h
 - Kalloc.c
 - Proc.c
 - Proc.h
 - Syscall.c
 - Syscall.h
 - Sysproc.c
- ~/user/ files:
 - User.h
 - Usys.pl
 - Test.c and lab1_test.c were created as test files for the syscalls
- Makefile

Change Explanations and Screenshots

Defs.h

```
110 void      print_hello(int);          // hello
111 int        sysinfo(int);              // sysinfo
112 int        procinfo(struct pinfo*);    // procinfo
```

Sysinfo and procinfo function headers were declared in defs.h

Kalloc.c

```
84 // Function to count the number of free pages
85 int
86 numFreePages(void)
87 {
88     // Declare variables
89     struct run *r;
90     int pageCount = 0;
91
92     // Loop through the freelist and count the number of pages
93     acquire(&kmem.lock);
94     for (r = kmem.freelist; r != 0; r = r->next) {
95         pageCount++;
96     }
97
98     // Return the count and release the lock
99     release(&kmem.lock);
100     return pageCount;
101 }
```

Function written to count the iterate over the list of free pages (kmem.freelist) and count each one

Proc.c

```
9 // Global variables
10 extern uint64 totalSyscalls;
11
12 // Outward functions
13 extern int numFreePages(void);
```

Global int to count totalSyscalls since boot declared

Global function header to count free pages (used in kalloc.c) declared

```

7  */
8  int
9  sysinfo(int param)
10 {
11     // Declare variables
12     struct proc *p = myproc();
13     struct proc *iter;
14     int numProcs = 0;
15
16     if (param == 0) {
17         // Count the number of processes
18         acquire(&p->lock);
19         for (iter = proc; iter < &proc[NPROC]; ++iter) {
20             if (iter->state != UNUSED) {
21                 numProcs++;
22             }
23         }
24         release(&p->lock);
25         return numProcs;
26     }
27     else if (param == 1) {
28         // Count the number of system calls
29         return totalSyscalls - 1;
30     }
31     else if (param == 2) {
32         // Count the number of free pages
33         return numFreePages();
34     }
35     else {
36         // Return that an error occurred
37         return -1;
38     }
39 }

```

Functionality of sysinfo implemented. If param == 0, sysinfo will iterate over all processes and return the number of processes that are either READY, RUNNING, WAITING, or ZOMBIE (or simply put, not UNUSED). If param == 1 then it will return the global variable, totalSyscalls - 1. The -1 offset is to account for the fact that sysinfo itself is a syscall that should not be self counted. If param == 2 then sysinfo calls the global function, numFreePages() and returns the number of free pages in memory. If param is anything else then sysinfo will return -1 as an error code.

```

0  int
1  procinfo(struct pinfo *in)
2  {
3      // Declare variables
4      struct proc *p = myproc();
5      struct pinfo temp;
6
7      // Check if the input is null
8      if (!in) {
9          return -1;
10     }
11
12     // Input the temp process info
13     temp.ppid = p->parent->pid;
14     temp.syscall_count = p->syscall_count - 1;
15     temp.page_usage = p->sz / PGSIZE;
16
17     // Copy the temp process info to the input process info
18     if (copyout(p->pagetable, (uint64)in, (char *)&temp, sizeof(temp)) < 0) {
19         return -1;
20     }
21
22     return 0;
23 }

```

Functionality of procinfo implemented. A proc pointer is declared to get the current process's PCB. A temp info struct is created since the kernel cannot write directly to the userspace info struct, in. If in is null then procinfo returns an error, otherwise procinfo will fill out the fields of temp based on data from the PCB from p. Once the data of temp is properly filled out, it is copied into the passed struct, in, using the copyout() function. This allows us to work around the fact that the kernel cannot write directly into the userspace memory.

Proc.h

```

108     };
109
110     struct pinfo {
111         int ppid;
112         int syscall_count;
113         int page_usage;
114     };

```

Pinfo struct declared to be filled out by the procinfo() syscall.

Syscall.c

```
10 // Global variables
11 uint64 totalSyscalls = 0;
```

Global variable to count syscalls initialized to 0 on bootup.

```
135 [SYS_hello] =sys_hello,      // hello: syscall entry
136 [SYS_sysinfo] =sys_sysinfo,  // sysinfo: syscall entry
137 [SYS_procinfo] =sys_procinfo, // procinfo: syscall entry
138 };
139
```

SYS_sysinfo and SYS_procinfo syscalls added to the entry table

```
146 num = p->trapframe->a7;
147 if (num > 0 && num < NELEM(syscalls) && syscalls[num]) {
148     // System call number is valid, increment totalSyscalls
149     totalSyscalls++;
150     p->syscall_count++;
151
152     // Use num to lookup the system call function for num, call it,
153     // and store its return value in p->trapframe->a0
154     p->trapframe->a0 = syscalls[num]();
155 }
```

Increment the global totalSyscalls and the calling process's syscall_count each time a successful is made.

Syscall.h

```
23 #define SYS_hello 22 // hello
24 #define SYS_sysinfo 23 // sysinfo
25 #define SYS_procinfo 24 // procinfo
26
```

Syscall numbers declared for sysinfo and procinfo

Sysproc.c

```
103 // sysinfo syscall definition
104 uint64
105 sys_sysinfo(void)
106 {
107     int n;
108     argint(0, &n);
109     return sysinfo(n);
110 }
111
112 // procinfo syscall definition
113 uint64
114 sys_procinfo(void)
115 {
116     struct pinfo *p;
117     argaddr(0, (uint64 *)&p);
118     return procinfo(p);
119 }
```

Sys_sysinfo and sys_procinfo functions implemented. In sys_sysinfo(), a integer is read from the command line and passed as a parameter to sysinfo() in proc.c. In sys_procinfo(), a pinfo struct pointer is declared and is pushed through argaddr() so that we can properly pass it to procinfo(). You cannot pass pointers as input to functions without this conversion via argaddr().

User.h

```
27 int sysinfo(int); // sysinfo
28 int procinfo(struct pinfo*); // procinfo
```

Function headers for sysinfo and procinfo declared in the userspace. The previous headers in defs.h were declared in the kernel space.

usys.pl

```
39  entry("hello"); # hello syscall for user
40  entry("sysinfo"); # sysinfo syscall for user
41  entry("procinfo"); # procinfo syscall for user
```

Lines 40 and 41 create wrapper functions in usys.S which will then call the appropriate system call. usys.S essentially acts as a medium between user and kernel space and usys.pl makes the user aware of the availability of the syscalls.

Makefile

```
135          $U/_test\
136          $U/_lab1_test\
```

Added these lines to the list of UPROGS for running the test files in qemu.

Lab1_test.c output

```
→Antheagao <~/xv6-riscv> Git:(↑ riscv) make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel
tio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ lab1_test 65535 2
[sysinfo] active proc: 3, syscalls: 51, free pages: 32564
[procinfo 4] ppid: 3, syscalls: 10, page usage: 20
[procinfo 5] ppid: 3, syscalls: 10, page usage: 20
[sysinfo] active proc: 5, syscalls: 243, free pages: 32510
$ lab1_test 65000 1
[sysinfo] active proc: 3, syscalls: 333, free pages: 32564
[procinfo 7] ppid: 6, syscalls: 10, page usage: 20
[sysinfo] active proc: 4, syscalls: 461, free pages: 32538
```

Description of XV6 Source Code

To start an info syscall, the user must perform a trap into the kernel space. When executing a trap, control is transferred to `trap.c` where it can determine that the user is performing a valid syscall. Once the validity is confirmed via syscall number (23 or 24), control is transferred to `syscall.c` which calls the correct function in `sysproc.c`. For 23 (`sysinfo`), it would call `sys_sysinfo`. Likewise, for 24 (`procinfo`), it would call `sys_procinfo`. The function in `sysproc.c` then calls the corresponding function in `proc.c` where the actual functionality of the syscall is implemented and returned. Once the syscall is done processing in `proc.c`, the return value is returned to the user via the trap frame initiated at the start of the syscall and control is given back to `syscall.c` which forwards control back to `trap.c`. `Trap.c` then restores the user's saved state before the syscall and returns control to the user at the point where he/she made the syscall.

Summary of Contributions

Jordan Kushner mainly contributed on part 1 and the report/video demonstration.

Jordan handled the initial setup and implemented the functionality of `sysinfo()`, as well as assisted on debugging part 2.

Anthony Mendez mainly contributed on part 2, implementing most of the functionality of the `procinfo()` syscall, as well as debugging part 1 and making some minor edits to the final report. A large majority of part 2 was implemented by Anthony.