

CS3500 LAB 2

Krutarth Patel EE23B137

14 August 2025

Brief

In this assignment we added two functionalities to the xv6-riscv kernel.

1. A trace utility like `strace` in Linux
2. A backtrace utility which prints the call stack

Details

0.1 Trace

Objective

When tracing is enabled for a process and a traced system call is called, this utility will print out the system call name and the status returned by the syscall after execution in the following format:

```
<pid>:  syscall <name> -> <return value>
```

Care is taken to copy the mask to the child process for each new process created through `fork()`.

Code

Makefile: A user program is added that can be run on the command line.

```
1 diff --git a/Makefile b/Makefile
2 index fab7bc9..d0d1f0a 100644
3 --- a/Makefile
4 +++ b/Makefile
5 @@ -142,6 +142,9 @@ UPROGS=\
6         $U/_grind\
7         $U/_wc\
8         $U/_zombie\
9 +        $U/_trace\
```

user/trace.c: This code is executed first upon running trace.

```
1 #include "kernel/types.h"
2 #include "user/user.h"
3
4 int main(int argc, char * argv[])
5 {
6     if(argc < 3){
7         fprintf(2, "Usage: trace <mask> <exec> <args>\n");
8         exit(1);
9     }
10    uint32 mask = atoi(argv[1]);
11    int ret = trace(mask);
12    if(ret < 0){
13        fprintf(2, "trace: failed! error code: %d\n", ret);
14        exit(1);
15    }
```

```

16     ret = exec(argv[2], &argv[2]);
17     if(ret < 0){
18         fprintf(2, "%s failed!\n", argv[2]);
19         exit(1);
20     }
21     exit(0);
22 }

```

First, it sets the mask provided by the user as command line argument by calling `trace(mask)`. Then the command to be traced is called using `exec(path, argv)`.

user/user.h: Function declaration.

```

1 diff --git a/user/user.h b/user/user.h
2 index f16fe27..bd3b7f0 100644
3 --- a/user/user.h
4 +++ b/user/user.h
5 @@ -22,6 +22,7 @@ int getpid(void);
6     char* sbrk(int);
7     int sleep(int);
8     int uptime(void);
9 +int trace(int);
10
11 // ulib.c
12 int stat(const char*, struct stat*);

```

user/usys.pl: To autogenerate stub for calling the underlying syscall.

```

1 diff --git a/user/usys.pl b/user/usys.pl
2 index 01e426e..9c97b05 100755
3 --- a/user/usys.pl
4 +++ b/user/usys.pl
5 @@ -36,3 +36,4 @@ entry("getpid");
6     entry("sbrk");
7     entry("sleep");
8     entry("uptime");
9 +entry("trace");

```

user/usys.S: The generated assembly stub.

```

1 trace:
2     li a7, SYS_trace
3     ecall
4     ret

```

kernel/syscall.h: Assigning the new trace syscall a number.

```

1 diff --git a/kernel/syscall.h b/kernel/syscall.h
2 index bc5f356..cc112b9 100644
3 --- a/kernel/syscall.h
4 +++ b/kernel/syscall.h
5 @@ -20,3 +20,4 @@
6     #define SYS_link    19
7     #define SYS_mkdir   20
8     #define SYS_close   21
9 +#define SYS_trace     22

```

kernel/proc.h: Adding member variable `trace_mask` to struct `proc`.

```

1 diff --git a/kernel/syscall.c b/kernel/syscall.c
2 diff --git a/kernel/proc.h b/kernel/proc.h
3 index d021857..ff50307 100644
4 --- a/kernel/proc.h
5 +++ b/kernel/proc.h
6 @@ -104,4 +104,5 @@ struct proc {

```

```

7   struct file *ofile[NOFILE]; // Open files
8   struct inode *cwd;          // Current directory
9   char name[16];              // Process name (debugging)
10  + uint32 trace_mask;
11  };

```

kernel/proc.c: Initializing the mask to 0 for a new process(in allocproc())and copying the mask to every child process(in fork()).

```

1  diff --git a/kernel/proc.c b/kernel/proc.c
2  index d280acf..89be5d2 100644
3  --- a/kernel/proc.c
4  +++ b/kernel/proc.c
5  @@ -146,6 +146,8 @@ found:
6      p->context.ra = (uint64)forkret;
7      p->context.sp = p->kstack + PGSIZE;
8
9  + // Set trace mask to 0
10 + p->trace_mask = 0;
11     return p;
12 }
13
14 @@ -320,6 +322,7 @@ fork(void)
15
16     acquire(&np->lock);
17     np->state = RUNNABLE;
18 + np->trace_mask = p->trace_mask;
19     release(&np->lock);
20
21     return pid;

```

kernel/sysproc.c: Implementing sys_trace().

```

1  uint64
2  sys_trace(void)
3  {
4      int mask;
5      argint(0, &mask);
6      struct proc * p = myproc();
7      acquire(&p->lock);
8      p->trace_mask = (uint32)mask;
9      release(&p->lock);
10     return 0;
11 }

```

NOTE: you need to get a mutex lock before updating the trace_mask member variable of struct proc. Since there can be race conditions where the current process is forked while sys_trace() is being executed.

kernel/syscall.c: Printing debug info for traced syscalls.

```

1  diff --git a/kernel/syscall.c b/kernel/syscall.c
2  index ed65409..22f1f9a 100644
3  --- a/kernel/syscall.c
4  +++ b/kernel/syscall.c
5  @@ -101,6 +101,35 @@ extern uint64 sys_unlink(void);
6      extern uint64 sys_link(void);
7      extern uint64 sys_mkdir(void);
8      extern uint64 sys_close(void);
9  +extern uint64 sys_trace(void);
10 +
11 +// An array mapping syscall numbers from syscall.h
12 +// to their names for trace()
13 +static char * syscall_names[] = {
14 +[SYS_fork]    = "fork",

```

```

15 +[SYS_exit]    = "exit",
16 +[SYS_wait]    = "wait",
17 +[SYS_pipe]    = "pipe",
18 +[SYS_read]    = "read",
19 +[SYS_kill]    = "kill",
20 +[SYS_exec]    = "exec",
21 +[SYS_fstat]   = "fstat",
22 +[SYS_chdir]   = "chdir",
23 +[SYS_dup]     = "dup",
24 +[SYS_getpid]  = "getpid",
25 +[SYS_sbrk]    = "sbrk",
26 +[SYS_sleep]   = "sleep",
27 +[SYS_uptime]  = "uptime",
28 +[SYS_open]    = "open",
29 +[SYS_write]   = "write",
30 +[SYS_mknod]   = "mknod",
31 +[SYS_unlink]  = "unlink",
32 +[SYS_link]    = "link",
33 +[SYS_mkdir]   = "mkdir",
34 +[SYS_close]   = "close",
35 +[SYS_trace]   = "trace",
36 +};
37 +
38
39 // An array mapping syscall numbers from syscall.h
40 // to the function that handles the system call.
41 @@ -126,6 +155,7 @@ static uint64 (*syscalls[])(void) = {
42 [SYS_link]    sys_link,
43 [SYS_mkdir]   sys_mkdir,
44 [SYS_close]   sys_close,
45 +[SYS_trace]   sys_trace,
46 };
47
48 void
49 @@ -139,6 +169,9 @@ syscall(void)
50     // Use num to lookup the system call function for num, call it,
51     // and store its return value in p->trapframe->a0
52     p->trapframe->a0 = syscalls[num]();
53 +    if((p->trace_mask & (1<<num))) {
54 +        printf("%d: syscall %s -> %lu\n", p->pid, syscall_names[num], p->trapframe->a0);
55 +    }
56     } else {
57         printf("%d %s: unknown sys call %d\n",
58             p->pid, p->name, num);

```

Proof

```

[krutarth@krutarth xv6-riscv]$ ostime spin
8.0
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 512M -smp 3 -nographic -global virtio-mmio,force-legacy=false -drive f
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ trace 32 grep hello README
3: syscall read -> 1023
3: syscall read -> 971
3: syscall read -> 298
3: syscall read -> 0
$ █

```

```

$ trace 2147483647 grep hello README
4: syscall trace -> 0
4: syscall exec -> 3
4: syscall open -> 3
4: syscall read -> 1023
4: syscall read -> 971
4: syscall read -> 298
4: syscall read -> 0
4: syscall close -> 0
$ trace 2 usertests forkforkfork
usertests starting
5: syscall fork -> 6
test forkforkfork: 5: syscall fork -> 7
7: syscall fork -> 8
backtrace:
800028ec
800027d4
80002554
8: syscall fork -> 9
8: syscall fork -> 10
9: syscall fork -> 11
8: syscall fork -> 12
8: syscall fork -> 13
8: syscall fork -> 14
8: syscall fork -> 15
8: syscall fork -> 16
8: syscall fork -> 17
8: syscall fork -> 18
8: syscall fork -> 19
8: syscall fork -> 20
8: syscall fork -> 21
8: syscall fork -> 22
8: syscall fork -> 23
9: syscall fork -> 24
8: syscall fork -> 25
9: syscall fork -> 26
8: syscall fork -> 27
8: syscall fork -> 28
8: syscall fork -> 29
8: syscall fork -> 30
8: syscall fork -> 31
8: syscall fork -> 32
8: syscall fork -> 33
8: syscall fork -> 34
8: syscall fork -> 35
8: syscall fork -> 36
8: syscall fork -> 37
8: syscall fork -> 38
8: syscall fork -> 39
8: syscall fork -> 40
10: syscall fork -> 41
8: syscall fork -> 42
9: syscall fork -> 43
9: syscall fork -> 44
9: syscall fork -> 45
9: syscall fork -> 46
9: syscall fork -> 47
9: syscall fork -> 48
9: syscall fork -> 49
9: syscall fork -> 50
8: syscall fork -> 51
8: syscall fork -> 52
8: syscall fork -> 53
8: syscall fork -> 54
8: syscall fork -> 55
9: syscall fork -> 56
9: syscall fork -> 57
9: syscall fork -> 58
9: syscall fork -> 59
8: syscall fork -> 60
8: syscall fork -> 61
8: syscall fork -> 62
8: syscall fork -> 63
8: syscall fork -> 64
8: syscall fork -> 65
8: syscall fork -> 66
8: syscall fork -> 67
8: syscall fork -> 18446744073709551615
backtrace:
800028ec
800027d4
80002554
OK
5: syscall fork -> 68
ALL TESTS PASSED
$ █

```

0.2 Backtrace

Objective

Calling `backtrace()` should print out the entire function call stack for the process up until that point in execution.

Code

kernel/defs.h: Function declaration

```
1 diff --git a/kernel/defs.h b/kernel/defs.h
2 index d1b6bb9..29f5a76 100644
3 --- a/kernel/defs.h
4 +++ b/kernel/defs.h
5 @@ -80,6 +80,7 @@ int pipewrite(struct pipe*, uint64, int);
6     int      printf(char*, ...) __attribute__((format(printf, 1, 2)));
7     void      panic(char*) __attribute__((noreturn));
8     void      printfinit(void);
9 +void        backtrace(void);
10
11 // proc.c
12 int          cpuid(void);
```

kernel/printf.c: Implementing `backtrace()`.

```
1 void
2 backtrace(void)
3 {
4     uint64 fp = r_fp();
5     uint64 top = PGROUNDUP(fp);
6     printf("backtrace:\n");
7     while(fp < top){
8         printf("%lx\n", *((uint64 *) (fp-8))); // The return address is at fp-8
9         fp = *((uint64 *) (fp - 16));         // The frame pointer to prev frame is at fp-16
10    }
11 }
```

Proof

```
[krutarth@krutarth xv6-riscv]$ ostime spin
[sudo] password for root:
8.0
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 512M -smp 3 -nographic -global virtio-mmio,force-legacy=false -drive file=
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ bttest
backtrace:
800028ec
800027d4
80002554
bttest: returned from sleep
$ █

[krutarth@krutarth xv6-riscv]$ riscv64-unknown-elf-addr2line -f -C -e kernel/kernel
800028ec
sys_sleep
/home/os-iitm/xv6-riscv/kernel/sysproc.c:58
800027d4
syscall
/home/os-iitm/xv6-riscv/kernel/syscall.c:171
80002554
usertrap
/home/os-iitm/xv6-riscv/kernel/trap.c:76
[krutarth@krutarth xv6-riscv]$ █
```