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.

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

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

```
#include "kernel/types.h"
#include "user/user.h"

int main(int argc, char * argv[])
{
  if(argc < 3){
    fprintf(2, "Usage: trace <mask> <exec> <args>\n");
    exit(1);
  }
  uint32 mask = atoi(argv[1]);
  int ret = trace(mask);
  if(ret < 0){
    fprintf(2, "trace: failed! error code: %d\n", ret);
    exit(1);
  }
  ret = exec(argv[2], &argv[2]);
  if(ret < 0){
    fprintf(2, "%s failed!\n", argv[2]);
    exit(1);
  }
  exit(0);
}
```

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.

```
diff --git a/user/user.h b/user/user.h
index f16fe27..bd3b7f0 100644
--- a/user/user.h
+++ b/user/user.h
@@ -22,6 +22,7 @@ int getpid(void);
 char* sbrk(int);
 int sleep(int);
 int uptime(void);
+int trace(int);

 // ulib.c
 int stat(const char*, struct stat*);
```

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

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

`user/usys.S`: The generated assembly stub.

```
trace:
 li a7, SYS_trace
 ecall
 ret
```

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

```
diff --git a/kernel/syscall.h b/kernel/syscall.h
index bc5f356..cc112b9 100644
--- a/kernel/syscall.h
+++ b/kernel/syscall.h
@@ -20,3 +20,4 @@
 #define SYS_link   19
 #define SYS_mkdir  20
 #define SYS_close  21
+#define SYS_trace  22
```

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

```
diff --git a/kernel/syscall.c b/kernel/syscall.c
diff --git a/kernel/proc.h b/kernel/proc.h
index d021857..ff50307 100644
--- a/kernel/proc.h
+++ b/kernel/proc.h
@@ -104,4 +104,5 @@ struct proc {
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
+  uint32 trace_mask;
 };
```

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

```
diff --git a/kernel/proc.c b/kernel/proc.c
index d280acf..89be5d2 100644
--- a/kernel/proc.c
+++ b/kernel/proc.c
@@ -146,6 +146,8 @@ found:
   p->context.ra = (uint64)forkret;
   p->context.sp = p->kstack + PGSIZE;

+  // Set trace mask to 0
+  p->trace_mask = 0;
   return p;
 }

@@ -320,6 +322,7 @@ fork(void)

   acquire(&np->lock);
   np->state = RUNNABLE;
+  np->trace_mask = p->trace_mask;
   release(&np->lock);

   return pid;
```

kernel/sysproc.c: Implementing sys_trace().

```
uint64
sys_trace(void)
{
  int mask;
  argint(0, &mask);
  struct proc * p = myproc();
  acquire(&p->lock);
  p->trace_mask = (uint32)mask;
  release(&p->lock);
  return 0;
}
```

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.

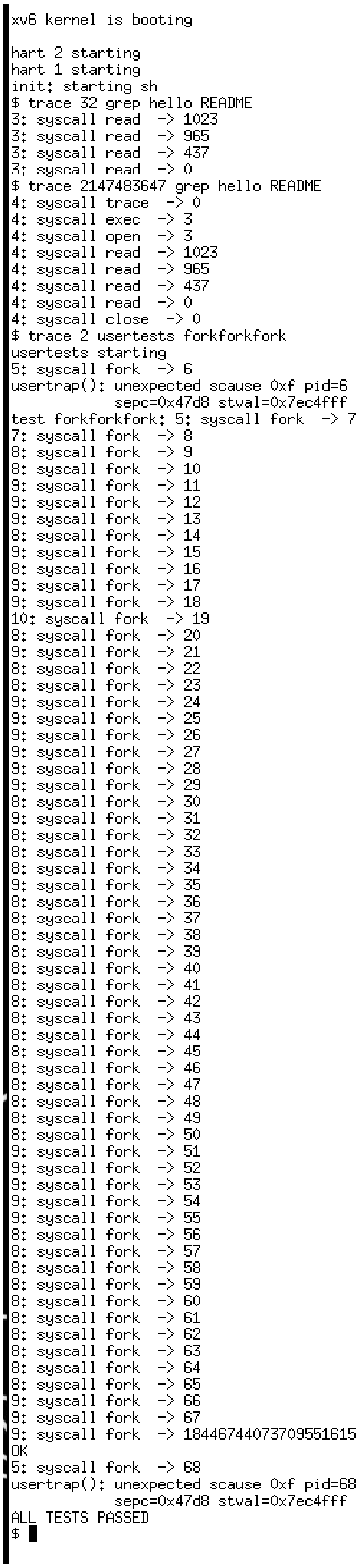
kernel/syscall.c: Printing debug info for traced syscalls.

```
diff --git a/kernel/syscall.c b/kernel/syscall.c
index ed65409..22f1f9a 100644
--- a/kernel/syscall.c
+++ b/kernel/syscall.c
@@ -101,6 +101,35 @@ extern uint64 sys_unlink(void);
 extern uint64 sys_link(void);
 extern uint64 sys_mkdir(void);
 extern uint64 sys_close(void);
+extern uint64 sys_trace(void);
+
+// An array mapping syscall numbers from syscall.h
+// to their names for trace()
+static char * syscall_names[] = {
+[SYS_fork]    = "fork",
+[SYS_exit]   = "exit",
+[SYS_wait]   = "wait",
+[SYS_pipe]   = "pipe",
+[SYS_read]   = "read",
+[SYS_kill]   = "kill",
+[SYS_exec]   = "exec",
+[SYS_fstat]  = "fstat",
+[SYS_chdir]  = "chdir",
+[SYS_dup]    = "dup",
+[SYS_getpid] = "getpid",
+[SYS_sbrk]   = "sbrk",
+[SYS_sleep]  = "sleep",
+[SYS_uptime] = "uptime",
+[SYS_open]   = "open",
+[SYS_write]  = "write",
+[SYS_mknod]  = "mknod",
+[SYS_unlink] = "unlink",
+[SYS_link]   = "link",
+[SYS_mkdir]  = "mkdir",
+[SYS_close]  = "close",
+[SYS_trace]  = "trace",
+};
+

 // An array mapping syscall numbers from syscall.h
 // to the function that handles the system call.
@@ -126,6 +155,7 @@ static uint64 (*syscalls[])(void) = {
 [SYS_link]    sys_link,
 [SYS_mkdir]   sys_mkdir,
 [SYS_close]   sys_close,
+[SYS_trace]   sys_trace,
 };

 void
@@ -139,6 +169,9 @@ syscall(void)
     // Use num to lookup the system call function for num, call it,
     // and store its return value in p->trapframe->a0
     p->trapframe->a0 = syscalls[num]();
+    if((p->trace_mask & (1<<num))){
+      printf("%d: syscall %s  -> %lu\n", p->pid, syscall_names[num], p->trapframe->a0);
+    }
   } else {
     printf("%d %s: unknown sys call %d\n",
             p->pid, p->name, num);
```

Proof



```
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  -> 965
3: syscall read  -> 437
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  -> 965
4: syscall read  -> 437
4: syscall read  -> 0
4: syscall close  -> 0
$ trace 2 usertests forkforkfork
usertests starting
5: syscall fork  -> 6
usertrap(): unexpected scause 0xf pid=6
            sepc=0x47d8 stval=0x7ec4fff
test forkforkfork: 5: syscall fork  -> 7
7: syscall fork  -> 8
8: syscall fork  -> 9
8: syscall fork  -> 10
9: syscall fork  -> 11
9: syscall fork  -> 12
9: syscall fork  -> 13
8: syscall fork  -> 14
9: syscall fork  -> 15
8: syscall fork  -> 16
9: syscall fork  -> 17
9: syscall fork  -> 18
10: syscall fork  -> 19
8: syscall fork  -> 20
9: syscall fork  -> 21
8: syscall fork  -> 22
8: syscall fork  -> 23
9: syscall fork  -> 24
9: syscall fork  -> 25
9: syscall fork  -> 26
9: syscall fork  -> 27
9: syscall fork  -> 28
9: syscall fork  -> 29
8: syscall fork  -> 30
9: syscall fork  -> 31
8: syscall fork  -> 32
8: syscall fork  -> 33
8: syscall fork  -> 34
9: syscall fork  -> 35
8: syscall fork  -> 36
8: syscall fork  -> 37
8: syscall fork  -> 38
8: syscall fork  -> 39
8: syscall fork  -> 40
8: syscall fork  -> 41
8: syscall fork  -> 42
8: syscall fork  -> 43
8: syscall fork  -> 44
8: syscall fork  -> 45
8: syscall fork  -> 46
8: syscall fork  -> 47
8: syscall fork  -> 48
8: syscall fork  -> 49
8: syscall fork  -> 50
9: syscall fork  -> 51
9: syscall fork  -> 52
9: syscall fork  -> 53
9: syscall fork  -> 54
9: syscall fork  -> 55
8: syscall fork  -> 56
8: syscall fork  -> 57
8: syscall fork  -> 58
8: 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
9: syscall fork  -> 66
9: syscall fork  -> 67
9: syscall fork  -> 18446744073709551615
OK
5: syscall fork  -> 68
usertrap(): unexpected scause 0xf pid=68
            sepc=0x47d8 stval=0x7ec4fff
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

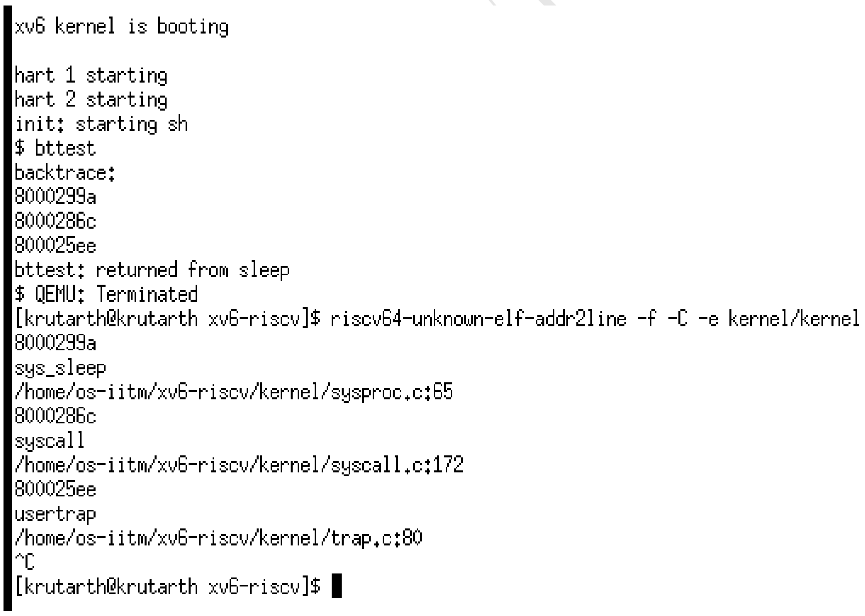
```
diff --git a/kernel/defs.h b/kernel/defs.h
index d1b6bb9..29f5a76 100644
--- a/kernel/defs.h
+++ b/kernel/defs.h
@@ -80,6 +80,7 @@ int             pipewrite(struct pipe*, uint64, int);
 int             printf(char*, ...) __attribute__ ((format (printf, 1, 2)));
 void            panic(char*) __attribute__((noreturn));
 void            printfinit(void);
+void            backtrace(void);

 // proc.c
 int             cpuid(void);
```

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

```c
void
backtrace(void)
{
  uint64 fp = r_fp();
  uint64 top = PGROUNDUP(fp);
  printf("backtrace:\n");
  while(fp < top){
    printf("%lx\n", *((uint64 *)(fp-8))); // The return address is at fp-8
    fp = *(uint64 *)(fp - 16);         // The frame pointer to prev frame is at fp-16
  }
}
```

Proof



```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ bttest
backtrace:
8000299a
8000286c
800025ee
bttest: returned from sleep
$ QEMU: Terminated
[krutarth@krutarth xv6-riscv]$ riscv64-unknown-elf-addr2line -f -C -e kernel/kernel
8000299a
sys_sleep
/home/os-iitm/xv6-riscv/kernel/sysproc.c:65
8000286c
syscall
/home/os-iitm/xv6-riscv/kernel/syscall.c:172
800025ee
usertrap
/home/os-iitm/xv6-riscv/kernel/trap.c:80
^C
[krutarth@krutarth xv6-riscv]$ 
```