Jami Tanskanen 612705

Kernel Hacking Summary: Adding getreadcount() to xv6

For my project, I added a new system call to the xv6 operating system. This system call, named getreadcount(), tells us how many times the read() function has been used.

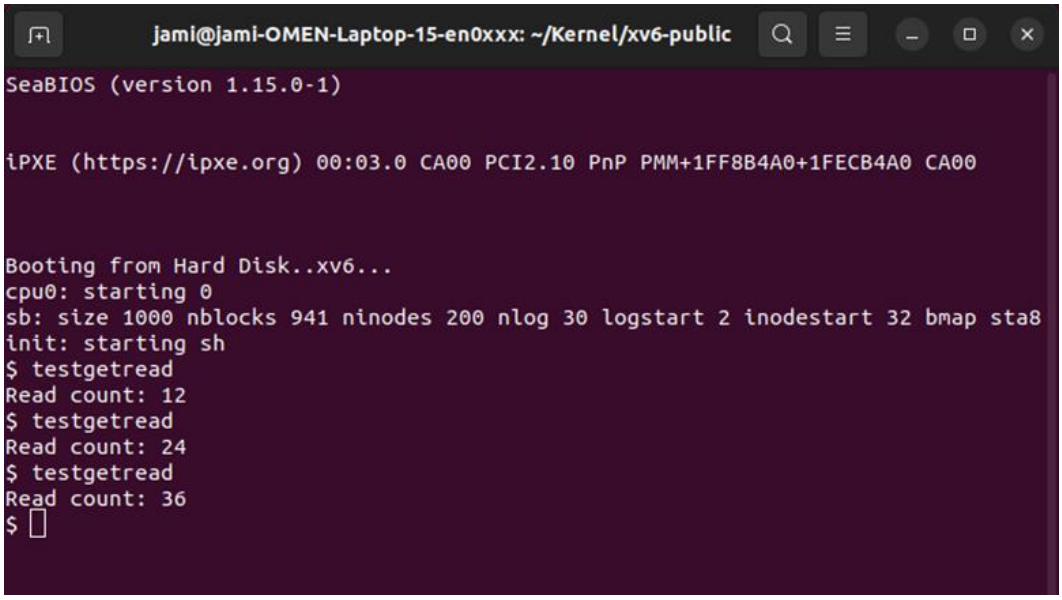
Steps I Took:

1. Changing read(): I added a counter in the sysfile.c file. Each time read() is used, this counter goes up by one.
2. Making getreadcount():
    - I listed the new system call in the defs.h file.
    - In the sysproc.c file, I made the actual function. It gives back the current counter number.
3. Adding to the System:
    - I gave getreadcount() its own number in syscall.h.
    - I updated syscall.c so it knows about the new function.
    - I made sure usys.S also knows about it.
4. Testing the New Function:
    - I wrote a test program, testgetread.c. It uses getreadcount() and shows its result.
    - I added testgetread to the Makefile.
    - I ran testgetread in xv6 to check if everything works (make qemu-nox).

Conclusion:

Now, xv6 has a new function that shows how many times read() has been used since booting.

Here's some screenshots:



jami@jami-OMEN-Laptop-15-en0xxx: ~/Kernel/xv6-public

GNU nano 6.2 testgetread.c

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

// User program to print the current read system call count
int main(void)
{
    printf(1, "Read count: %d\n", getreadcount());
    exit();
}
```

jami@jami-OMEN-Laptop-15-en0xxx: ~/Kernel/xv6-public

GNU nano 6.2 syscall.c

```c
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getreadcount(void); // Function prototype for get readcount system call

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_getreadcount] sys_getreadcount, // Map getreadcount system call number to its function
};
```

GNU nano 6.2 syscall.h

```c
// 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_getreadcount 22
```

jami@jami-OMEN-Laptop-15-en0xxx: ~/Kernel/xv6-public

GNU nano 6.2 usys.S

```
  .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(getreadcount) // User-facing stub for the getreadcount system call
```

jami@jami-OMEN-Laptop-15-en0xxx: ~/Kernel/xv6-public

GNU nano 6.2 sysproc.c

```
      release(&tickslock);
      return -1;
    }
    sleep(&ticks, &tickslock);
  }
  release(&tickslock);
  return 0;
}

// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
  uint xticks;

  acquire(&tickslock);
  xticks = ticks;
  release(&tickslock);
  return xticks;
}

// Returns the total number of read() system calls made since boot
int sys_getreadcount(void)
{
    return readcount;
}
```

jami@jami-OMEN-Laptop-15-en0xxx: ~/Kernel/xv6-public

GNU nano 6.2 sysfile.c

```
  if(argfd(0, 0, &f) < 0)
    return -1;
  if((fd=fdalloc(f)) < 0)
    return -1;
  filedup(f);
  return fd;
}

int
sys_read(void)
{
  readcount++; // Incrementing the global readcount to track the number of read() system calls made
  struct file *f;
  int n;
  char *p;

  if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
    return -1;
  return fileread(f, p, n);
}
```

```
  GNU nano 6.2                                          main.c
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"

// This variable is a global counter to keep track of the
//  number of read() system calls made
int readcount = 0;
static void startothers(void);
static void mpmain(void)  __attribute__((noreturn));
extern pde_t *kpgdir;
extern char end[]; // first address after kernel loaded from ELF file

// Bootstrap processor starts running C code here.
// Allocate a real stack and switch to it, first
// doing some setup required for memory allocator to work.
int
main(void)
{
  readcount = 0; // Initialize the read() system call counter to zero when the Kernel starts
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc();      // kernel page table
```