Xv6 System Calls: Implementation Guide

Xv6 Kernel Development Team

May 2025

Contents

1 Introduction

This document provides a detailed guide for implementing five system calls in the Xv6 operating system: `getinterruptcount`, `getdiskstats`, `getprocinfo`, `getsysinfo`, and `top`. Each system call enhances Xv6's functionality by providing user-space programs access to kernel-level information, such as interrupt counts, disk statistics, process details, system resource usage, and top CPU-consuming processes. The implementations include kernel modifications, system call handlers, and user-space interfaces.

2 System Call: getinterruptcount

2.1 Overview

The `getinterruptcount` system call returns the total number of hardware interrupts handled by the Xv6 kernel since boot, useful for monitoring kernel activity or debugging.

2.2 Implementation Steps

1. **Declare Global Interrupt Counter**: Add a global variable in `kernel/trap.c` to track interrupts.
2. **Increment Counter in Interrupt Handler**: Modify `kerneltrap` to increment the counter for each interrupt.
3. **Create Kernel Function**: Define `get_interrupt_count` to return the counter value.
4. **Add System Call Interface**: Implement the system call handler and register it.
5. **User-Space Integration**: Add user-space declarations and stubs.

2.3 Kernel Code

Listing 1: kernel/trap.c: Global Counter and Handler

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

uint64 interrupt_count = 0;

void kerneltrap() {
  // Existing code ...
  if ((which_dev = devintr()) == 0) {
    panic("kerneltrap");
  }
  interrupt_count++;  // Increment for each recognized interrupt
  // Existing code ...
}

uint64 get_interrupt_count(void) {
  return interrupt_count;
}
```

Listing 2: kernel/sysproc.c: System Call Handler

```c
uint64 sys_getinterruptcount(void) {
  return get_interrupt_count();
}
```

Listing 3: kernel/defs.h: Function Prototype

```c
uint64 get_interrupt_count(void);
```

2.4 System Call Registration

Listing 4: kernel/syscall.h: Syscall Number

```c
#define SYS_getinterruptcount 22
```

Listing 5: kernel/syscall.c: Syscall Table

```c
extern uint64 sys_getinterruptcount(void);
[SYS_getinterruptcount] sys_getinterruptcount,
```

Listing 6: user/usys.pl: User Stub

```perl
entry("getinterruptcount")
```

Listing 7: user/user.h: User Declaration

```c
uint64 getinterruptcount(void);
```

2.5 User Program Example

Listing 8: user/interrupt.c: Test Program

```c
#include "user.h"
#include "stdio.h"
int main() {
  uint64 count = getinterruptcount();
  printf("Total interrupts: %llu\n", count);
  return 0;
}
```

3 System Call: getdiskstats

3.1 Overview

The `getdiskstats` system call retrieves disk I/O statistics, including read/write counts and bytes, from the `virtio` disk driver.

3.2 Implementation Steps

1. **Define Disk Stats Structure**: Create `struct diskstats` in `kernel/virtio.h`.
2. **Add Global Counters**: Track statistics in `kernel/virtio`$_{d}isk.c.$`Implement Kernel Function` $: Copy stats to user space in$ `kernel/sysproc.c`.
3. **Register System Call**: Add syscall number and handler.
4. **User-Space Wrapper**: Provide user-space declarations and test program.

3.3 Kernel Code

Listing 9: kernel/virtio.h: Disk Stats Structure

```
struct diskstats {
  uint64 read_count;
  uint64 write_count;
  uint64 read_bytes;
  uint64 write_bytes;
};
```

Listing 10: kernel/virtio$_{d}isk.c : GlobalCounters$

```
#include "virtio.h"
struct diskstats disk_stats = {0, 0, 0, 0};

// In read completion:
disk_stats.read_count++;
disk_stats.read_bytes += number_of_bytes_read;

// In write completion:
disk_stats.write_count++;
disk_stats.write_bytes += number_of_bytes_written;
```

Listing 11: kernel/sysproc.c: System Call Handler

```
#include "virtio.h"
#include "defs.h"
#include "proc.h"

uint64 sys_getdiskstats(void) {
  struct diskstats stats = disk_stats;
  struct diskstats *user_ptr;
  argaddr(0, (uint64*)&user_ptr);
  if (copyout(myproc()->pagetable, (uint64)user_ptr, (char *)&stats,
      sizeof(stats)) < 0)
    return -1;
  return 0;
}
```

3.4 System Call Registration

Listing 12: kernel/syscall.h: Syscall Number

```
#define SYS_getdiskstats 23
```

Listing 13: kernel/syscall.c: Syscall Table

```
extern uint64 sys_getdiskstats(void);
[SYS_getdiskstats] sys_getdiskstats,
```

Listing 14: user/usys.pl: User Stub

```
entry("getdiskstats")
```

Listing 15: user/user.h: User Declaration

```
struct diskstats {
  uint64 read_count;
```

```
  uint64 write_count;
  uint64 read_bytes;
  uint64 write_bytes;
};
int getdiskstats(struct diskstats *stats);
```

3.5 User Program Example

Listing 16: user/diskinfo.c: Test Program

```
#include "user.h"
#include "diskstats.h"
#include "stdio.h"
int main(void) {
  struct diskstats stats;
  if (getdiskstats(&stats) == 0) {
    printf("Disk reads: %d\n", stats.read_count);
    printf("Disk writes: %d\n", stats.write_count);
    printf("Bytes read: %d\n", stats.read_bytes);
    printf("Bytes written: %d\n", stats.write_bytes);
  } else {
    printf("Failed to get disk stats\n");
  }
  return 0;
}
```

4 System Call: getprocinfo

4.1 Overview

The `getprocinfo` system call provides information about a specific process, given its PID, including state, parent PID, memory size, and name.

4.2 Implementation Steps

1. **Define procinfo Structure**: Create `struct procinfo` in `kernel/sysinfo.h`.
2. **Kernel Function**: Implement `getprocinfo` in `kernel/proc.c`.
3. **System Call Wrapper**: Add handler in `kernel/sysproc.c`.
4. **Register System Call**: Add syscall number and mappings.
5. **User-Space Test**: Provide user-space program to test the call.

4.3 Kernel Code

Listing 17: kernel/sysinfo.h: procinfo Structure

```
#ifndef SYSINFO_H
#define SYSINFO_H
struct procinfo {
  int pid;
  int state;
  int ppid;
  uint64 sz;
  char name[16];
};
```

```c
#endif
```

Listing 18: kernel/proc.c: Kernel Function

```c
#include "sysinfo.h"
uint64 getprocinfo(int pid, struct procinfo *info) {
  struct proc *p;
  for (p = proc; p < &proc[NPROC]; p++) {
    if (p->state != UNUSED && p->pid == pid) {
      info->pid = p->pid;
      info->state = p->state;
      info->ppid = p->parent ? p->parent->pid : 0;
      info->sz = p->sz;
      safestrcpy(info->name, p->name, sizeof(p->name));
      return 0;
    }
  }
  return -1;
}
```

Listing 19: kernel/sysproc.c: System Call Handler

```c
#include "sysinfo.h"
uint64 sys_getprocinfo(void) {
  int pid;
  uint64 user_addr;
  struct procinfo info;
  argint(0, &pid);
  argaddr(1, &user_addr);
  if (getprocinfo(pid, &info) < 0)
    return -1;
  if (copyout(myproc()->pagetable, user_addr, (char *)&info, sizeof(
     info)) < 0)
    return -1;
  return 0;
}
```

4.4 System Call Registration

Listing 20: kernel/syscall.h: Syscall Number

```c
#define SYS_getprocinfo 24
```

Listing 21: kernel/syscall.c: Syscall Table

```c
extern uint64 sys_getprocinfo(void);
[SYS_getprocinfo] sys_getprocinfo,
```

Listing 22: user/usys.pl: User Stub

```perl
entry("getprocinfo")
```

Listing 23: user/user.h: User Declaration

```c
#include "sysinfo.h"
int getprocinfo(int pid, struct procinfo *info);
```

4.5 User Program Example

Listing 24: user/sysinfo.c: Test Program

```c
#include "kernel/types.h"
#include "user/user.h"
int main(int argc, char *argv[]) {
  struct procinfo info;
  int pid = 1;
  if (getprocinfo(pid, &info) == 0) {
    printf("PID: %d, State: %d, PPID: %d, Size: %d, Name: %s\n",
           info.pid, info.state, info.ppid, info.sz, info.name);
  } else {
    printf("Process %d not found\n", pid);
  }
  exit(0);
}
```

5 System Call: getsysinfo

5.1 Overview

The `getsysinfo` system call returns system-wide information, including the number of processes, runnable processes, and free memory in bytes.

5.2 Implementation Steps

1. **Define sysinfo Structure**: Create `struct sysinfo` in `kernel/sysinfo.h`.
2. **Count Free Pages**: Add `freepages` in `kernel/kalloc.c`.
3. **Implement System Call**: Add handler in `kernel/sysproc.c`.
4. **Register System Call**: Add syscall number and mappings.
5. **User Program**: Create a test program.

5.3 Kernel Code

Listing 25: kernel/sysinfo.h: sysinfo Structure

```c
#ifndef SYSINFO_H
#define SYSINFO_H
struct sysinfo {
  int nprocs;
  int nrunnable;
  int freemem;
};
#endif
```

Listing 26: kernel/kalloc.c: Free Pages Counter

```c
int freepages(void) {
  struct run *r;
  int count = 0;
  acquire(&kmem.lock);
  for (r = kmem.freelist; r; r = r->next)
    count++;
  release(&kmem.lock);
```

```
  return count;
}
```

Listing 27: kernel/sysproc.c: System Call Handler

```
#include "sysinfo.h"
uint64 sys_getsysinfo(void) {
  struct sysinfo info;
  struct proc *p;
  uint64 addr;
  info.nprocs = 0;
  info.nrunnable = 0;
  info.freemem = freepages() * PGSIZE;
  for (p = proc; p < &proc[NPROC]; p++) {
    if (p->state != UNUSED)
      info.nprocs++;
    if (p->state == RUNNABLE)
      info.nrunnable++;
  }
  argaddr(0, &addr);
  if (copyout(myproc()->pagetable, addr, (char *)&info, sizeof(info)) <
     0)
    return -1;
  return 0;
}
```

5.4 System Call Registration

Listing 28: kernel/syscall.h: Syscall Number

```
#define SYS_getsysinfo 25
```

Listing 29: kernel/syscall.c: Syscall Table

```
extern uint64 sys_getsysinfo(void);
[SYS_getsysinfo] sys_getsysinfo,
```

Listing 30: user/usys.pl: User Stub

```
entry("getsysinfo")
```

Listing 31: user/user.h: User Declaration

```
#include "../kernel/sysinfo.h"
int getsysinfo(struct sysinfo *info);
```

5.5 User Program Example

Listing 32: user/sysinfo.c: Test Program

```
#include "user.h"
#include "../kernel/sysinfo.h"
int main(void) {
  struct sysinfo info;
  if (getsysinfo(&info) < 0) {
    printf("sysinfo call failed\n");
    exit(1);
  }
```

```c
  printf("Processes: %d\n", info.nprocs);
  printf("Runnable: %d\n", info.nrunnable);
  printf("Free mem: %d bytes\n", info.freemem);
  exit(0);
}
```

6 System Call: top

6.1 Overview

The `top` system call returns information about the top CPU-consuming processes, including PID, CPU time, and name, sorted by CPU usage.

6.2 Implementation Steps

1. **Define top_proc Structure**: Add `struct top_proc` and `cputime` field in `kernel/proc.h`.
2. **Track CPU Time**: Modify the scheduler in `kernel/proc.c`.
3. **Implement gettop Function**: Add sorting logic in `kernel/proc.c`.
4. **System Call Handler**: Implement `sys_top` in `kernel/sysproc.c`.
5. **Register System Call**: Add syscall number and mappings.
6. **User Program**: Create a test program.

6.3 Kernel Code

Listing 33: kernel/proc.h: top_proc Structure

```c
struct top_proc {
  int pid;
  uint cputime;
  char name[16];
};
struct proc {
  // ...
  uint cputime;  // Total CPU time used
  // ...
};
int gettop(struct top_proc *tops, int n);
```

Listing 34: kernel/proc.c: Scheduler and gettop

```c
void scheduler(void) {
  struct proc *p;
  for(;;) {
    sti();
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
      if(p->state != RUNNABLE)
        continue;
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&cpu->scheduler, proc->context);
      switchkvm();
```

```c
      p->cputime++;   // Increment CPU time
      proc = 0;
    }
    release(&ptable.lock);
  }
}

int gettop(struct top_proc *tops, int n) {
  struct proc *p;
  int i, j;
  acquire(&ptable.lock);
  for (i = 0; i < n; i++) {
    tops[i].pid = 0;
    tops[i].cputime = 0;
    memset(tops[i].name, 0, sizeof(tops[i].name));
  }
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if(p->state == UNUSED)
      continue;
    for(i = 0; i < n; i++) {
      if(p->cputime > tops[i].cputime) {
        for(j = n - 1; j > i; j--) {
          tops[j] = tops[j-1];
        }
        tops[i].pid = p->pid;
        tops[i].cputime = p->cputime;
        safestrcpy(tops[i].name, p->name, sizeof(tops[i].name));
        break;
      }
    }
  }
  release(&ptable.lock);
  return n;
}
```

Listing 35: kernel/sysproc.c: System Call Handler

```c
int sys_top(void) {
  struct top_proc *user_tops;
  int n;
  if (argptr(0, (void*)&user_tops, sizeof(struct top_proc) * 64) < 0 ||
      argint(1, &n) < 0) {
    return -1;
  }
  struct top_proc kernel_tops[n];
  int count = gettop(kernel_tops, n);
  if (copyout(myproc()->pagetable, (uint64)user_tops, (char*)
     kernel_tops, sizeof(struct top_proc) * count) < 0) {
    return -1;
  }
  return count;
}
```

6.4 System Call Registration

Listing 36: kernel/syscall.h: Syscall Number

```c
#define SYS_top 26
```

Listing 37: kernel/syscall.c: Syscall Table

```
extern uint64 sys_top(void);
[SYS_top] sys_top,
```

Listing 38: user/usys.pl: User Stub

```
entry("top")
```

Listing 39: user/user.h: User Declaration

```
struct top_proc {
  int pid;
  unsigned int cputime;
  char name[16];
};
int top(struct top_proc *tops, int n);
```

6.5 User Program Example

Listing 40: user/top.c: Test Program

```
#include "user.h"
#include "fcntl.h"
#include "types.h"
#include "stat.h"
int main(int argc, char *argv[]) {
  struct top_proc tops[10];
  int n = 10;
  int count = top(tops, n);
  if (count < 0) {
    printf("top syscall failed\n");
    exit(1);
  }
  printf("Top %d processes by CPU time:\n", count);
  for (int i = 0; i < count; i++) {
    printf("PID: %d, CPU Time: %d, Name: %s\n", tops[i].pid, tops[i].
        cputime, tops[i].name);
  }
  exit(0);
}
```

7 Makefile Modifications

To compile the user programs, update `Makefile` to include the new executables:

Listing 41: Makefile: User Programs

```
UPROGS=\
  _cat\
  _echo\
  _forktest\
  _interrupt\
  _diskinfo\
  _sysinfo\
  _top\
```

8 Conclusion

This document detailed the implementation of five system calls in Xv6: `getinterruptcount`, `getdiskstats`, `getprocinfo`, `getsysinfo`, and `top`. Each system call was implemented with kernel modifications, system call handlers, and user-space test programs, enabling robust system monitoring and introspection.