Bryan Arnold

4/20/18

CSE 4300

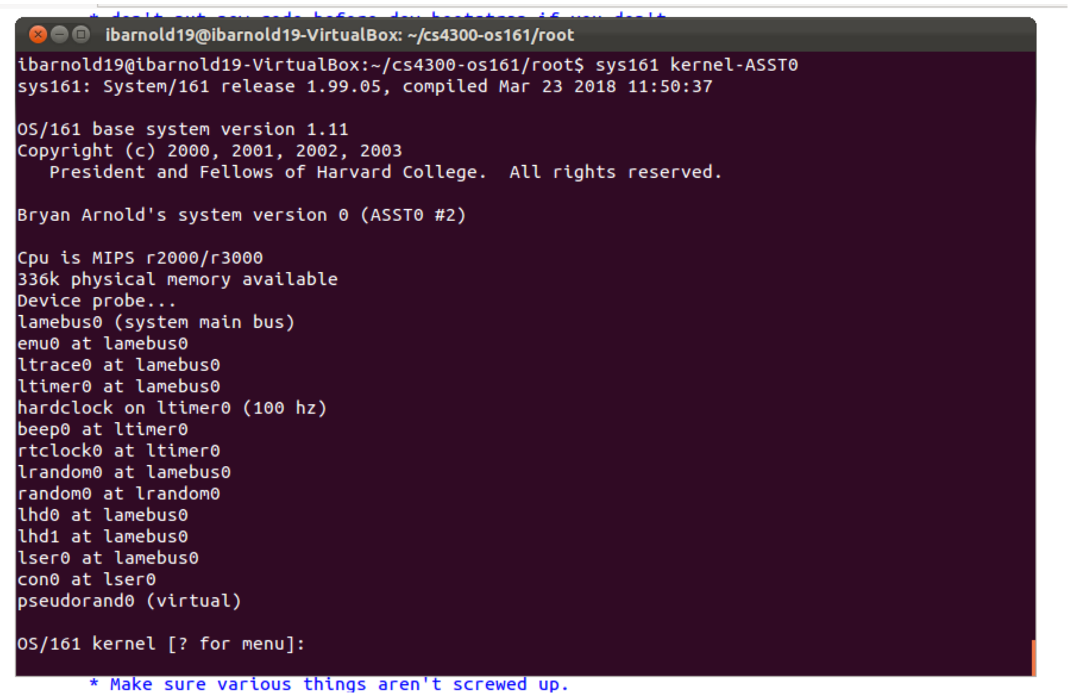
Programming Assignment 3

Work I Completed

For this assignment, I half completed every component of the assignment. I understood that I needed to implement each thing inside the syscall.c method and all header files associated with it, but I kept getting undefined reference errors on my newly created methods. I looked back to how reboot() is fully implemented, and even went as far as to fully copy everywhere that reboot() is associated for _exit() as well. I kept getting errors, so I decided to try and bypass some of these errors by doing implementations inside menu.c, since it is closely associated to execution of commands. I got them to work as separate from the syscall.c way, whilst still trying to implement there, so I partially completed them. Both my test files do work. So, part A was completed, B-D are partially finished, and parts E-F are complete. I will provide all source code of my changes at the end.

PART A

For this part, I simply went into main.c and changed the kprintf() statement that had the section to include the name of the system owner. Here is the output:

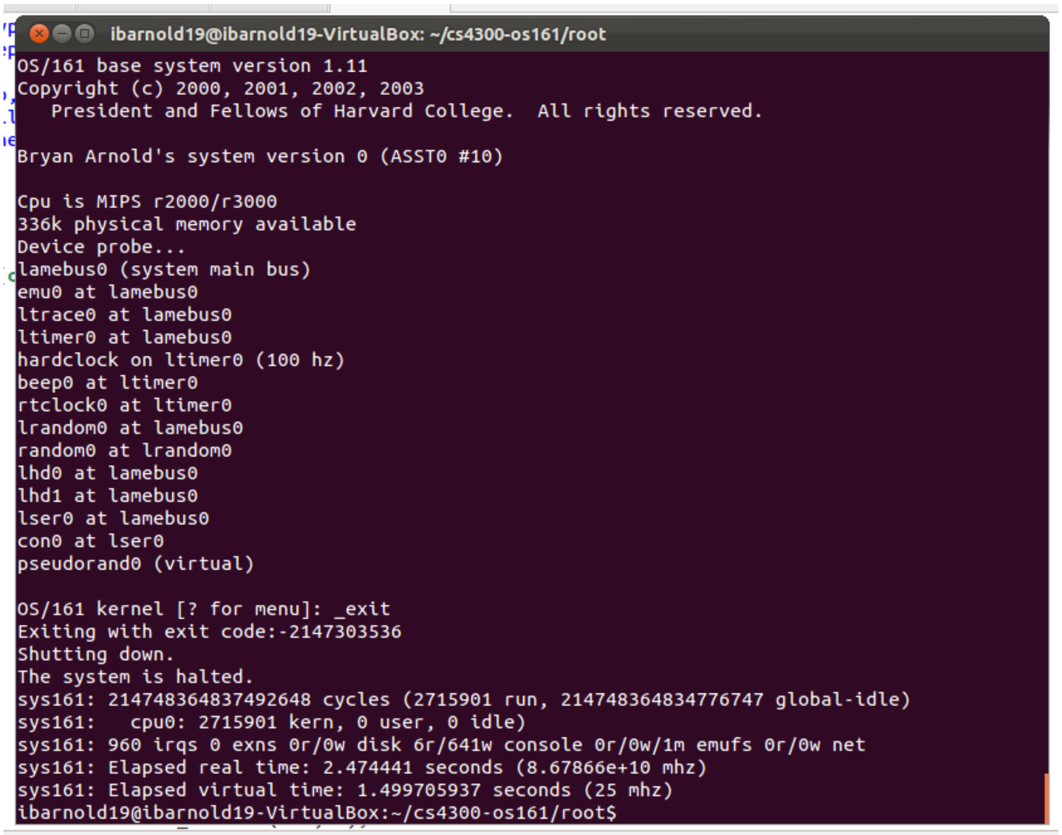


```
ibarnold19@ibarnold19-VirtualBox:~/cs4300-os161/root$ sys161 kernel-ASST0
sys161: System/161 release 1.99.05, compiled Mar 23 2018 11:50:37

OS/161 base system version 1.11
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #2)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]:
```

```
       * Make sure various things aren't screwed up.
```

PART B

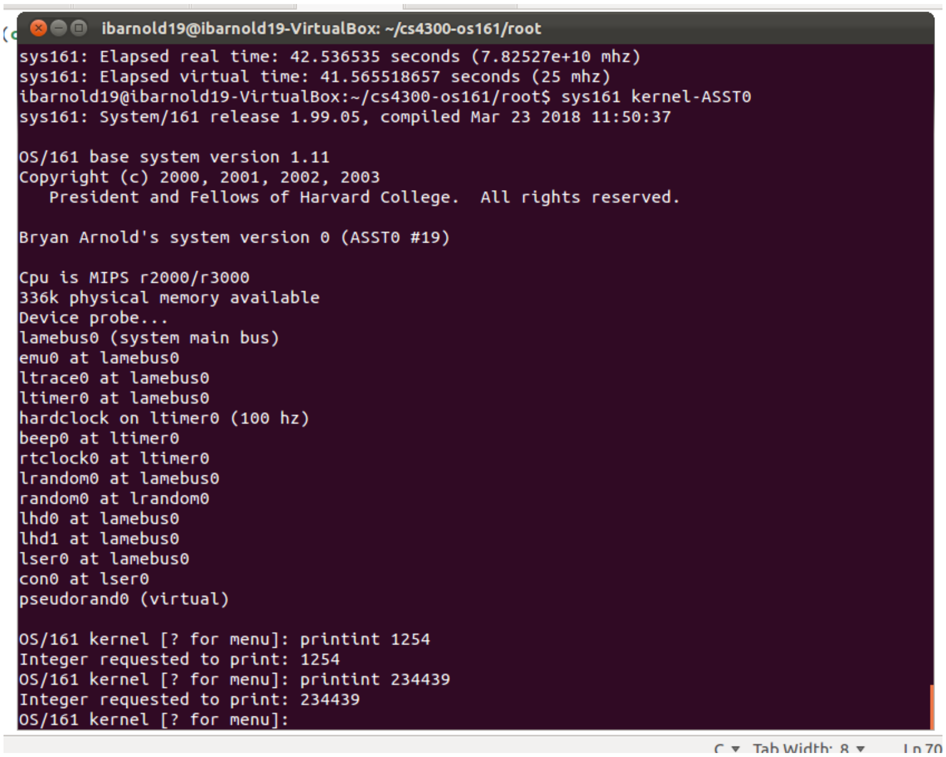
For this part, I tried to implement it just as reboot() is. I went into syscall.c, put in a case for SYS__exit, this calls _exit() function, since I couldn't get sys__exit() to work properly. I put this header of _exit() into syscall.h, and implemented it in menu.c, since I couldn't get it to work in it own file for some reason in userprog. Here is a test:



```
OS/161 base system version 1.11
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #10)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]: _exit
Exiting with exit code:-2147303536
Shutting down.
The system is halted.
sys161: 214748364837492648 cycles (2715901 run, 214748364834776747 global-idle)
sys161:   cpu0: 2715901 kern, 0 user, 0 idle)
sys161: 960 irqs 0 exns 0r/0w disk 6r/641w console 0r/0w/1m emufs 0r/0w net
sys161: Elapsed real time: 2.474441 seconds (8.67866e+10 mhz)
sys161: Elapsed virtual time: 1.499705937 seconds (25 mhz)
ibarnold19@ibarnold19-VirtualBox:~/cs4300-os161/root$
```

Part C

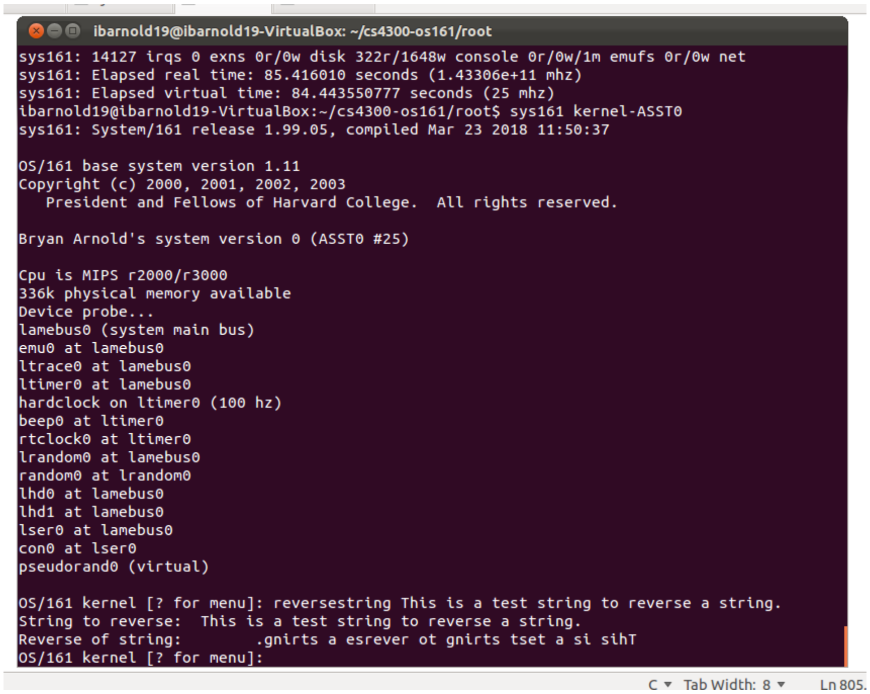
For this, I did the same steps as _exit() but called it SYS_printint instead and simply called kprintf() on an integer in syscall.c. Next, I put the header in syscall.h, implemented it in menu.c for int printint(int toPrint), and added a macro in callno.h for the new syscall of 32. Here is a test:



```
sys161: Elapsed real time: 42.536535 seconds (7.82527e+10 mhz)
sys161: Elapsed virtual time: 41.565518657 seconds (25 mhz)
ibarnold19@ibarnold19-VirtualBox:~/cs4300-os161/root$ sys161 kernel-ASST0
sys161: System/161 release 1.99.05, compiled Mar 23 2018 11:50:37

OS/161 base system version 1.11
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #19)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]: printint 1254
Integer requested to print: 1254
OS/161 kernel [? for menu]: printint 234439
Integer requested to print: 234439
OS/161 kernel [? for menu]:
```

PART D

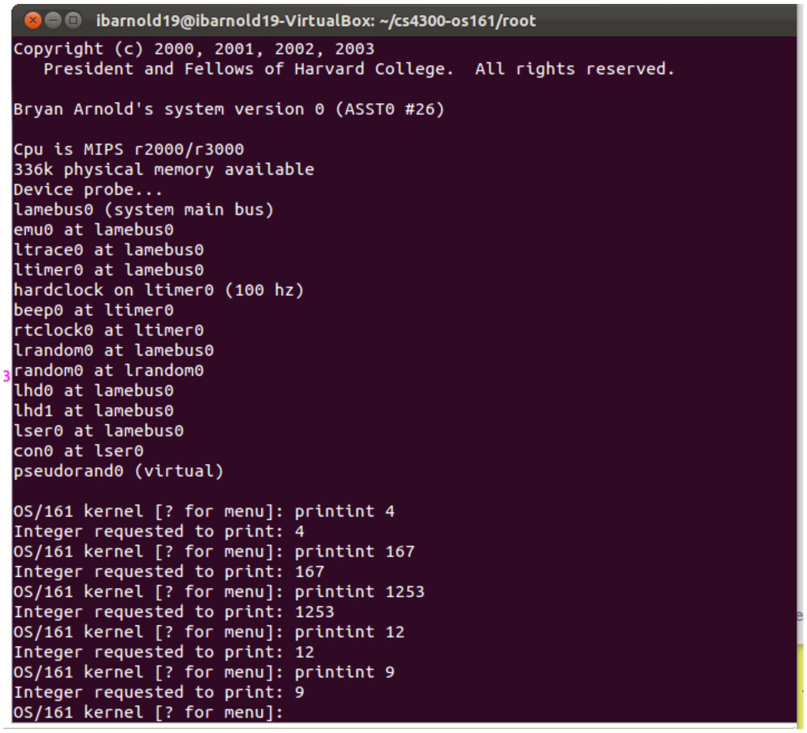
For this, I did the same steps as printint() but called it SYS_reverse instead and simply called kprintf() on an integer in syscall.c. Next, I put the header in syscall.h, implemented it in menu.c for int reversestring(cont char* str, len), and added a macro in callno.h for the new syscall of 32. Here is a test:


ibarnold19@ibarnold19-VirtualBox: ~/cs4300-os161/root
C ▾ Tab Width: 8 ▾ Ln 805,

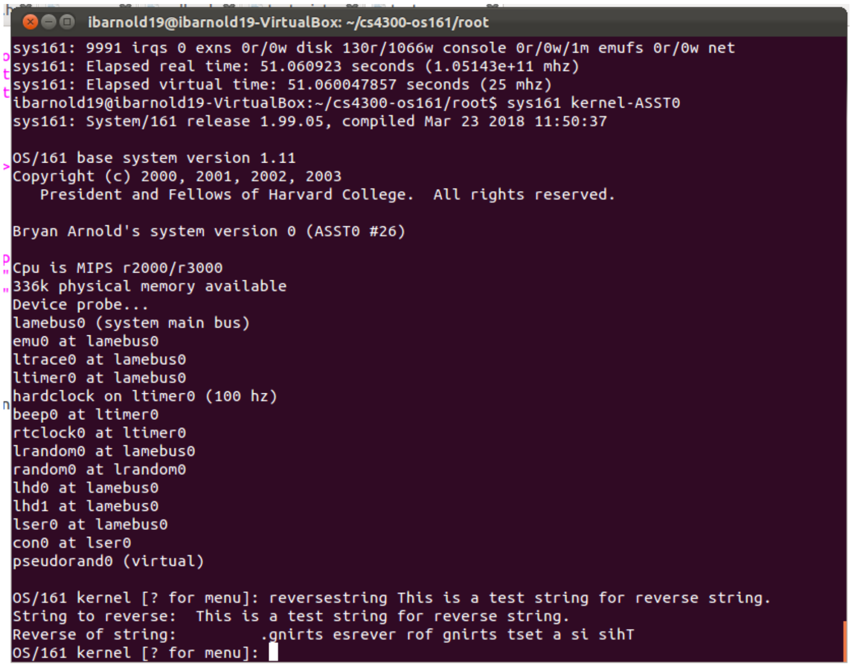
```
sys161: 14127 irqs 0 exns 0r/0w disk 322r/1648w console 0r/0w/1m emufs 0r/0w net
sys161: Elapsed real time: 85.416010 seconds (1.43306e+11 mhz)
sys161: Elapsed virtual time: 84.443550777 seconds (25 mhz)
ibarnold19@ibarnold19-VirtualBox:~/cs4300-os161/root$ sys161 kernel-ASST0
sys161: System/161 release 1.99.05, compiled Mar 23 2018 11:50:37

OS/161 base system version 1.11
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #25)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]: reversestring This is a test string to reverse a string.
String to reverse:  This is a test string to reverse a string.
Reverse of string:        .gnirts a esrever ot gnirts tset a si sihT
OS/161 kernel [? for menu]:
```

PART E-F

This is just testing the created syscalls. I tried putting them in testbin and having them used using p testbin/testprint, but it wouldn't work. So, I just did it more manually. Here are some tests:



```
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #26)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]: printint 4
Integer requested to print: 4
OS/161 kernel [? for menu]: printint 167
Integer requested to print: 167
OS/161 kernel [? for menu]: printint 1253
Integer requested to print: 1253
OS/161 kernel [? for menu]: printint 12
Integer requested to print: 12
OS/161 kernel [? for menu]: printint 9
Integer requested to print: 9
OS/161 kernel [? for menu]:
```



```
sys161: 9991 irqs 0 exns 0r/0w disk 130r/1066w console 0r/0w/1m emufs 0r/0w net
sys161: Elapsed real time: 51.060923 seconds (1.05143e+11 mhz)
sys161: Elapsed virtual time: 51.060047857 seconds (25 mhz)
ibarnold19@ibarnold19-VirtualBox:~/cs4300-os161/root$ sys161 kernel-ASST0
sys161: System/161 release 1.99.05, compiled Mar 23 2018 11:50:37

OS/161 base system version 1.11
Copyright (c) 2000, 2001, 2002, 2003
   President and Fellows of Harvard College.  All rights reserved.

Bryan Arnold's system version 0 (ASST0 #26)

Cpu is MIPS r2000/r3000
336k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
hardclock on ltimer0 (100 hz)
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
pseudorand0 (virtual)

OS/161 kernel [? for menu]: reversestring This is a test string for reverse string.
String to reverse:  This is a test string for reverse string.
Reverse of string:         .gnirts esrever rof gnirts tset a si sihT
OS/161 kernel [? for menu]: 
```

SOURCE CODE

Here is all the source code of every file I changed in the os161-1.11 folder.

callno.h

```
#ifndef _KERN_CALLNO_H_
#define _KERN_CALLNO_H_

/*
 * System call numbers.
 * Caution: this file is parsed by a shell script to generate the
assembly
 * language system call stubs. Don't add weird stuff between the
markers.
 */

/*CALLBEGIN*/
#define SYS__exit        0
#define SYS_execv        1
#define SYS_fork         2
#define SYS_waitpid      3
#define SYS_open         4
#define SYS_read         5
#define SYS_write        6
#define SYS_close        7
#define SYS_reboot       8
#define SYS_sync         9
#define SYS_sbrk         10
#define SYS_getpid       11
#define SYS_ioctl        12
#define SYS_lseek        13
#define SYS_fsync        14
#define SYS_ftruncate    15
#define SYS_fstat        16
#define SYS_remove       17
#define SYS_rename       18
#define SYS_link         19
#define SYS_mkdir        20
#define SYS_rmdir        21
#define SYS_chdir        22
#define SYS_getdirentry  23
#define SYS_symlink      24
#define SYS_readlink     25
#define SYS_dup2         26
#define SYS_pipe         27
#define SYS___time       28
#define SYS___getcwd     29
#define SYS_stat         30
#define SYS_lstat        31
#define SYS_printint     32
#define SYS_reverse      33
```

```
/*CALLEND*/


#endif /* _KERN_CALLNO_H_ */
```

menu.c

```
/*
 * In-kernel menu and command dispatcher.
 */

#include <types.h>
#include <kern/errno.h>
#include <kern/unistd.h>
#include <kern/limits.h>
#include <lib.h>
#include <clock.h>
#include <thread.h>
#include <syscall.h>
#include <uio.h>
#include <vfs.h>
#include <sfs.h>
#include <test.h>
#include "opt-synchprobs.h"
#include "opt-sfs.h"
#include "opt-net.h"

#define _PATH_SHELL "/bin/sh"

#define MAXMENUARGS  16

void
getinterval(time_t s1, u_int32_t ns1, time_t s2, u_int32_t ns2,
          time_t *rs, u_int32_t *rns)
{
      if (ns2 < ns1) {
           ns2 += 1000000000;
           s2--;
      }

      *rns = ns2 - ns1;
      *rs = s2 - s1;
}

////////////////////////////////////////////////////////////
//
// Command menu functions

/*
 * Function for a thread that runs an arbitrary userlevel program by
 * name.
 *
 * Note: this cannot pass arguments to the program. You may wish to
 * change it so it can, because that will make testing much easier
 * in the future.
 *
 * It copies the program name because runprogram destroys the copy
 * it gets by passing it to vfs_open().
 */
static
void
cmd_progthread(void *ptr, unsigned long nargs)
{
      char **args = ptr;
      char progname[128];
      int result;

      assert(nargs >= 1);

      if (nargs > 2) {
            kprintf("Warning: argument passing from menu not
supported\n");
      }

      /* Hope we fit. */
      assert(strlen(args[0]) < sizeof(progname));

      strcpy(progname, args[0]);

      result = runprogram(progname);
      if (result) {
            kprintf("Running program %s failed: %s\n", args[0],
                  strerror(result));
            return;
      }

      /* NOTREACHED: runprogram only returns on error. */
}

/*
 * Common code for cmd_prog and cmd_shell.
 *
 * This function uses the one_thread_only() function to make
 * the kernel menu thread wait until the newly-launched program
 * has finished.  The one_thread_only() function is a bit ugly
 * (it works in this specific situation but not more generally)
 * Once you have A2 working, you should be able to use your
 * call your waitpid implementation (instead of one_thread_only())
 * to provide the necessary synchronization.
 *
 * Also note that because the subprogram's thread uses the "args"
 * array and strings, there will be a race condition between the
 * subprogram and the menu input code if the menu thread is not
 * made to wait (using one_thread_only or some other mechanism)
 */
static
int
common_prog(int nargs, char **args)
{
      int result;

#if OPT_SYNCHPROBS
      kprintf("Warning: this probably won't work with a "
           "synchronization-problems kernel.\n");
#endif

      result = thread_fork(args[0] /* thread name */,
                 args /* thread arg */, nargs /* thread arg */,
                 cmd_progthread, NULL);
      if (result) {
           kprintf("thread_fork failed: %s\n", strerror(result));
           return result;
      }

      /* this function is a bit of a hack that is used to make
       * the kernel menu thread wait until the newly-forked
         * thread completes before the menu thread returns */
      while (!one_thread_only()) {
        clocksleep(1);
      }

      return 0;
}

/*
 * Command for running an arbitrary userlevel program.
 */
static
int
cmd_prog(int nargs, char **args)
{
      if (nargs < 2) {
           kprintf("Usage: p program [arguments]\n");
           return EINVAL;
      }

      /* drop the leading "p" */
      args++;
      nargs--;

      return common_prog(nargs, args);
}

/*
```

```
 * Command for starting the system shell.
 */
static
int
cmd_shell(int nargs, char **args)
{
        (void)args;
        if (nargs != 1) {
                kprintf("Usage: s\n");
                return EINVAL;
        }

        args[0] = (char *)_PATH_SHELL;

        return common_prog(nargs, args);
}

/*
 * Command for changing directory.
 */
static
int
cmd_chdir(int nargs, char **args)
{
        if (nargs != 2) {
                kprintf("Usage: cd directory\n");
                return EINVAL;
        }

        return vfs_chdir(args[1]);
}

/*
 * Command for printing the current directory.
 */
static
int
cmd_pwd(int nargs, char **args)
{
        char buf[PATH_MAX+1];
        struct uio ku;
        int result;

        (void)nargs;
        (void)args;

        mk_kuio(&ku, buf, sizeof(buf)-1, 0, UIO_READ);
        result = vfs_getcwd(&ku);
        if (result) {
                kprintf("vfs_getcwd failed (%s)\n", strerror(result));
                return result;
        }
	/* null terminate */
	buf[sizeof(buf)-1-ku.uio_resid] = 0;

	/* print it */
	kprintf("%s\n", buf);

	return 0;
}

/*
 * Command for running sync.
 */
static
int
cmd_sync(int nargs, char **args)
{
	(void)nargs;
	(void)args;

	vfs_sync();

	return 0;
}

/*
 * Command for doing an intentional panic.
 */
static
int
cmd_panic(int nargs, char **args)
{
	(void)nargs;
	(void)args;

	panic("User requested panic\n");
	return 0;
}

/*
 * Command for shutting down.
 */
static
int
cmd_quit(int nargs, char **args)
{
	(void)nargs;
	(void)args;

	vfs_sync();
	sys_reboot(RB_POWEROFF);
	thread_exit();
```

```
        return 0;
}

/*
 * Command for mounting a filesystem.
 */

/* Table of mountable filesystem types. */
static const struct {
        const char *name;
        int (*func)(const char *device);
} mounttable[] = {
#if OPT_SFS
        { "sfs", sfs_mount },
#endif
        { NULL, NULL }
};

static
int
cmd_mount(int nargs, char **args)
{
        char *fstype;
        char *device;
        int i;

        if (nargs != 3) {
                kprintf("Usage: mount fstype device:\n");
                return EINVAL;
        }

        fstype = args[1];
        device = args[2];

        /* Allow (but do not require) colon after device name */
        if (device[strlen(device)-1]==':') {
                device[strlen(device)-1] = 0;
        }

        for (i=0; mounttable[i].name; i++) {
                if (!strcmp(mounttable[i].name, fstype)) {
                        return mounttable[i].func(device);
                }
        }
        kprintf("Unknown filesystem type %s\n", fstype);
        return EINVAL;
}

static
int
cmd_unmount(int nargs, char **args)
{
        char *device;

        if (nargs != 2) {
                kprintf("Usage: unmount device:\n");
                return EINVAL;
        }

        device = args[1];

        /* Allow (but do not require) colon after device name */
        if (device[strlen(device)-1]==':') {
                device[strlen(device)-1] = 0;
        }

        return vfs_unmount(device);
}

/*
 * Command to set the "boot fs".
 *
 * The boot filesystem is the one that pathnames like /bin/sh with
 * leading slashes refer to.
 *
 * The default bootfs is "emu0".
 */
static
int
cmd_bootfs(int nargs, char **args)
{
        char *device;

        if (nargs != 2) {
                kprintf("Usage: bootfs device\n");
                return EINVAL;
        }

        device = args[1];

        /* Allow (but do not require) colon after device name */
        if (device[strlen(device)-1]==':') {
                device[strlen(device)-1] = 0;
        }

        return vfs_setbootfs(device);
}

static
int
cmd_kheapstats(int nargs, char **args)
{
        (void)nargs;
        (void)args;
	kheap_printstats();

	return 0;
}

////////////////////////////////////////
//
// Menus.

static
void
showmenu(const char *name, const char *x[])
{
	int ct, half, i;

	kprintf("\n");
	kprintf("%s\n", name);

	for (i=ct=0; x[i]; i++) {
		ct++;
	}
	half = (ct+1)/2;

	for (i=0; i<half; i++) {
		kprintf("    %-36s", x[i]);
		if (i+half < ct) {
			kprintf("%s", x[i+half]);
		}
		kprintf("\n");
	}

	kprintf("\n");
}

static const char *opsmenu[] = {
	"[s]       Shell                     ",
	"[p]       Other program             ",
	"[mount]   Mount a filesystem        ",
	"[unmount] Unmount a filesystem      ",
	"[bootfs]  Set \"boot\" filesystem     ",
	"[pf]      Print a file              ",
	"[cd]      Change directory          ",
	"[pwd]     Print current directory   ",
	"[sync]    Sync filesystems          ",
	"[panic]   Intentional panic         ",
	"[q]       Quit and shut down        ",
	NULL
};

static
int
```

```
cmd_opsmenu(int n, char **a)
{
	(void)n;
	(void)a;

	showmenu("OS/161 operations menu", opsmenu);
	return 0;
}

static const char *testmenu[] = {
	"[at]  Array test                    ",
	"[bt]  Bitmap test                   ",
	"[qt]  Queue test                    ",
	"[km1] Kernel malloc test            ",
	"[km2] kmalloc stress test           ",
	"[tt1] Thread test 1                 ",
	"[tt2] Thread test 2                 ",
	"[tt3] Thread test 3                 ",
#if OPT_NET
	"[net] Network test                  ",
#endif
	"[sy1] Semaphore test                ",
	"[sy2] Lock test             (1)     ",
	"[sy3] CV test               (1)     ",
	"[fs1] Filesystem test               ",
	"[fs2] FS read stress        (4)     ",
	"[fs3] FS write stress       (4)     ",
	"[fs4] FS write stress 2     (4)     ",
	"[fs5] FS create stress      (4)     ",
	NULL
};

static
int
cmd_testmenu(int n, char **a)
{
	(void)n;
	(void)a;

	showmenu("OS/161 tests menu", testmenu);
	kprintf("    (1) These tests will fail until you finish the "
		"synch assignment.\n");
	kprintf("    (4) These tests will fail until you finish the "
		"file system assignment.\n");
	kprintf("\n");

	return 0;
}

static const char *mainmenu[] = {
	"[?o] Operations menu                ",
	"[?t] Tests menu                     ",
#if OPT_SYNCHPROBS
      /*   "[1a] Cat/mouse with semaphores      ", */
      /*      "[1b] Cat/mouse with locks and CVs   ", */
      "[1a] Cat/mouse                      ",
      "[1b] Stoplight                      ",
#endif
      "[kh] Kernel heap stats              ",
      "[q] Quit and shut down              ",
      NULL
};

static
int
cmd_mainmenu(int n, char **a)
{
      (void)n;
      (void)a;

      showmenu("OS/161 kernel menu", mainmenu);
      return 0;
}

////////////////////////////////////////
//
// Command table.

static struct {
      const char *name;
      int (*func)(int nargs, char **args);
} cmdtable[] = {
      /* menus */
      { "?",           cmd_mainmenu },
      { "h",           cmd_mainmenu },
      { "help",  cmd_mainmenu },
      { "?o",          cmd_opsmenu },
      { "?t",          cmd_testmenu },

      /* operations */
      { "s",           cmd_shell },
      { "p",           cmd_prog },
      { "mount", cmd_mount },
      { "unmount",     cmd_unmount },
      { "bootfs",      cmd_bootfs },
      { "pf",          printfile },
      { "cd",          cmd_chdir },
      { "pwd",   cmd_pwd },
      { "sync",  cmd_sync },
      { "panic", cmd_panic },
      { "q",           cmd_quit },
      { "exit",  cmd_quit },
      { "halt",  cmd_quit },
#if OPT_SYNCHPROBS
	/* in-kernel synchronization problems */
	/* { "1a",	catmousesem }, */
	{ "1a",		catmouse},
	/* { "1c",	createcars }, */
	{ "1b",		createcars },
#endif

	/* stats */
	{ "kh",		cmd_kheapstats },

	/* base system tests */
	{ "at",		arraytest },
	{ "bt",		bitmaptest },
	{ "qt",		queuetest },
	{ "km1",	malloctest },
	{ "km2",	mallocstress },
#if OPT_NET
	{ "net",	nettest },
#endif
	{ "tt1",	threadtest },
	{ "tt2",	threadtest2 },
	{ "tt3",	threadtest3 },
	{ "sy1",	semtest },

	/* synchronization assignment tests */
	{ "sy2",	locktest },
	{ "sy3",	cvtest },

	/* file system assignment tests */
	{ "fs1",	fstest },
	{ "fs2",	readstress },
	{ "fs3",	writestress },
	{ "fs4",	writestress2 },
	{ "fs5",	createstress },

	{ NULL, NULL }
};

/*
 * Process a single command.
 */
static
int
cmd_dispatch(char *cmd)
{
	time_t beforesecs, aftersecs, secs;
	u_int32_t beforensecs, afternsecs, nsecs;
	char *args[MAXMENUARGS];
	int nargs=0;
	char *word;
	char *context;
```

```
      int i, result;

      for (word = strtok_r(cmd, " \t", &context);
           word != NULL;
           word = strtok_r(NULL, " \t", &context)) {

           if (nargs >= MAXMENUARGS) {
                kprintf("Command line has too many words\n");
                return E2BIG;
           }
           args[nargs++] = word;
      }

      if (nargs==0) {
           return 0;
      }

      for (i=0; cmdtable[i].name; i++) {
           if (*cmdtable[i].name && !strcmp(args[0],
cmdtable[i].name)) {
                assert(cmdtable[i].func!=NULL);

                gettime(&beforesecs, &beforensecs);

                result = cmdtable[i].func(nargs, args);

                gettime(&aftersecs, &afternsecs);
                getinterval(beforesecs, beforensecs,
                         aftersecs, afternsecs,
                         &secs, &nsecs);

                kprintf("Operation took %lu.%09lu seconds\n",
                      (unsigned long) secs,
                      (unsigned long) nsecs);

                return result;
           }
      }

      kprintf("%s: Command not found\n", args[0]);
      return EINVAL;
}

/*
 * Evaluate a command line that may contain multiple semicolon-
delimited
 * commands.
 *
 * If "isargs" is set, we're doing command-line processing; print the
 * comamnds as we execute them and panic if the command is invalid or
fails.
 */
```

```
static
void
menu_execute(char *line, int isargs)
{
      char *command;
      char *context;
      int result;

      for (command = strtok_r(line, ";", &context);
           command != NULL;
           command = strtok_r(NULL, ";", &context)) {

           if (isargs) {
                 kprintf("OS/161 kernel: %s\n", command);
           }

           result = cmd_dispatch(command);
           if (result) {
                 kprintf("Menu command failed: %s\n",
strerror(result));
                 if (isargs) {
                      panic("Failure processing kernel arguments\n");
                 }
           }
      }
}

int strncmp(char* s1, char* s2, int n){

      int i = 0;

      for(i = 0; i < n; i++){

           if(s1[i] != s2[i]){

                 return 1;

           }

      }

      return 0;

}

/* Implementation of the _exit() function */

void _exit(int exitCode){

      kprintf("Exiting with exit code:%d\n", exitCode);
      vfs_sync();
      sys_reboot(RB_POWEROFF);
```

```
    thread_exit();

}

int printInt(int toPrint){

    kprintf("Integer requested to print: %d\n", toPrint);

    if(toPrint % 3 == 0) {

        return 0;

    } else {

        return 1;

    }

}

int acquireInt(char* str){

    int result = 0;
    int sign = 1;
    int i = 0;

    if(str[1] == '-' || str[2] == '-'){

        sign = -1;

    }

    for(; str[i] != '\0'; ++i){

        if(str[i] == ' ' || str[i] == NULL){

            continue;

        } else {

            result = result * 10 + str[i] - '0';

        }

    }

    return sign * result;

}

int reversestring(const char* str, int len){
	int i = 0;

	kprintf("String to reverse: %s\n", str);
	kprintf("Reverse of string: ");

	for(i = len - 1; i >= 0; i--){

		if(str[i] == NULL){

			continue;

		}

		kprintf("%c", str[i]);

	}

	kprintf("\n");

	if(len % 5 == 0){

		return 0;

	} else {

		return 1;

	}

}

/*
 * Command menu main loop.
 *
 * First, handle arguments passed on the kernel's command line from
 * the bootloader. Then loop prompting for commands.
 *
 * The line passed in from the bootloader is treated as if it had been
 * typed at the prompt. Semicolons separate commands; spaces and tabs
 * separate words (command names and arguments).
 *
 * So, for instance, to mount an SFS on lhd0 and make it the boot
 * filesystem, and then boot directly into the shell, one would use
 * the kernel command line
 *
 *      "mount sfs lhd0; bootfs lhd0; s"
 */

void
menu(char *args)
{
	char buf[64];
```

```
        int i = 0;
        char rest[59];

        menu_execute(args, 1);

        while (1) {

                kprintf("OS/161 kernel [? for menu]: ");
                kgets(buf, sizeof(buf));

                for(i = 0; i < 59; i++){

                        rest[i] = buf[i + 5];

                }

                if(strncmp(buf, "_exit", 5) == 0){

                        _exit(rest);

                }

                for(i = 0; i < 56; i++){

                        rest[i] = buf[i + 8];

                }

                if(strncmp(buf, "printint", 8) == 0){

                        int a = acquireInt(rest);
                        printInt(a);
                        continue;

                }

                for(i = 0; i < 51; i++){

                        rest[i] = buf[i + 13];

                }

                if(strncmp(buf, "reversestring", 13) == 0){

                        reversestring(rest, 51);
                        continue;

                }

                menu_execute(buf, 0);
        }
}
```

mips-crt0.S

```
/*
 * crt0.o for MIPS r2000/r3000.
 *
 * crt stands for "C runtime".
 *
 * Basically, this is the startup code that gets invoked before
main(),
 * and regains control when main returns.
 *
 * All we really do is save a copy of argv for use by the err* and
warn*
 * functions, and call exit when main returns.
 */

#include <machine/asmdefs.h>
#include <kern/callno.h>

      .set noreorder  /* so we can use delay slots explicitly */

      .text
      .globl __start
      .type __start,@function
      .ent __start
__start:
      /* Load the "global pointer" register */
      la gp, _gp

      /*
       * We expect that the kernel passes argc in a0 and argv in a1.
       * We do not expect the kernel to set up a complete stack frame,
       * however.
       *
       * The MIPS ABI decrees that every caller will leave 16 bytes of
       * space in the bottom of its stack frame for writing back the
       * values of a0-a3, even when calling functions that take fewer
       * than four arguments. It also requires the stack to be aligned
       * to an 8-byte boundary. (This is because of 64-bit MIPS, which
       * we're not dealing with... but we'll conform to the standard.)
       */
      li t0, 0xfffffff8          /* mask for stack alignment */
      and sp, sp, t0             /* align the stack */
      addiu sp, sp, -16          /* create our frame */

      sw a1, __argv   /* save second arg (argv) in __argv for use later
*/

      jal main   /* call main */
      nop        /* delay slot */
```

```
     /*
      * Now, we have the return value of main in v0.
      *
      * Move it to s0 (which is callee-save) so we still have
      * it in case exit() returns.
      *
      * Also move it to a0 so it's the argument to exit.
      */
     move s0, v0      /* save return value */
     jal exit   /* call exit() */
     move a0, s0      /* Set argument (in delay slot) */

     /*
      * If we got here, something is broken in exit().
      * Try using _exit().
      */
     jal _exit  /* Try _exit() */
     move a0, s0      /* Set argument (in delay slot) */

     /*
      * If *that* doesn't work, try doing an _exit syscall by hand.
      */
1:
     move a0, s0
     li v0, SYS__exit
     syscall

     /*
      * ...and if we still can't exit, there's not much we can do
      * but keep trying.
      */
     j 1b       /* loop back */
     nop        /* delay slot */
     .end __start
```

syscall.c

```
#include <types.h>
#include <kern/errno.h>
#include <lib.h>
#include <machine/pcb.h>
#include <machine/spl.h>
#include <machine/trapframe.h>
#include <kern/callno.h>
#include <syscall.h>


/*
 * System call handler.
 *
 * A pointer to the trapframe created during exception entry (in
 * exception.S) is passed in.
 *
 * The calling conventions for syscalls are as follows: Like ordinary
 * function calls, the first 4 32-bit arguments are passed in the 4
 * argument registers a0-a3. In addition, the system call number is
 * passed in the v0 register.
 *
 * On successful return, the return value is passed back in the v0
 * register, like an ordinary function call, and the a3 register is
 * also set to 0 to indicate success.
 *
 * On an error return, the error code is passed back in the v0
 * register, and the a3 register is set to 1 to indicate failure.
 * (Userlevel code takes care of storing the error code in errno and
 * returning the value -1 from the actual userlevel syscall function.
 * See src/lib/libc/syscalls.S and related files.)
 *
 * Upon syscall return the program counter stored in the trapframe
 * must be incremented by one instruction; otherwise the exception
 * return code will restart the "syscall" instruction and the system
 * call will repeat forever.
 *
 * Since none of the OS/161 system calls have more than 4 arguments,
 * there should be no need to fetch additional arguments from the
 * user-level stack.
 *
 * Watch out: if you make system calls that have 64-bit quantities as
 * arguments, they will get passed in pairs of registers, and not
 * necessarily in the way you expect. We recommend you don't do it.
 * (In fact, we recommend you don't use 64-bit quantities at all. See
 * arch/mips/include/types.h.)
 */

void
mips_syscall(struct trapframe *tf)
{
        int callno;
        int32_t retval;
        int err;

        assert(curspl==0);

        callno = tf->tf_v0;

        /*
         * Initialize retval to 0. Many of the system calls don't
         * really return a value, just 0 for success and -1 on
         * error. Since retval is the value returned on success,
         * initialize it to 0 by default; thus it's not necessary to
         * deal with it except for calls that return other values,
 * like write.
 */

retval = 0;

switch (callno) {

    case SYS_reboot:
      err = sys_reboot(tf->tf_a0);
      break;

    case SYS__exit:

      err = _exit(tf->tf_a0);
      break;

    case SYS_printint:

      err = 0;
      kprintf("%d", tf->tf_a0);
      break;

    case SYS_reverse:

      err = 0;
      kprintf("%s", tf->tf_a0);
      break;

    default:

      kprintf("Unknown syscall %d\n", callno);
      err = ENOSYS;
      break;

}


if (err) {
      /*
       * Return the error code. This gets converted at
       * userlevel to a return value of -1 and the error
       * code in errno.
       */
      tf->tf_v0 = err;
      tf->tf_a3 = 1;      /* signal an error */
}
else {
      /* Success. */
      tf->tf_v0 = retval;
      tf->tf_a3 = 0;      /* signal no error */
}
```

```
      /*
       * Now, advance the program counter, to avoid restarting
       * the syscall over and over again.
       */

      tf->tf_epc += 4;

      /* Make sure the syscall code didn't forget to lower spl */
      assert(curspl==0);
}

void
md_forkentry(struct trapframe *tf)
{
      /*
       * This function is provided as a reminder. You need to write
       * both it and the code that calls it.
       *
       * Thus, you can trash it and do things another way if you
prefer.
       */

      (void)tf;
}
```

syscall.h

```
#ifndef _SYSCALL_H_
#define _SYSCALL_H_

/*
 * Prototypes for IN-KERNEL entry points for system call
implementations.
 */

int sys_reboot(int code);
int sys__exit(int code);
int sys_printchar(int toPrint);
int sys_reversestring(const char* str, int len);


#endif /* _SYSCALL_H_ */
```

testprint.c

```
#include <types.h>
#include <kern/errno.h>
```

```
#include <kern/unistd.h>
#include <kern/limits.h>
#include <lib.h>
#include <clock.h>
#include <thread.h>
#include <syscall.h>
#include <uio.h>
#include <vfs.h>
#include <sfs.h>
#include <test.h>
#include "opt-synchprobs.h"
#include "opt-sfs.h"
#include "opt-net.h"

int main(){

      int test[5] = {4, 167, 1253, 12, 9};
      int i = 0;

      for(i = 0; i < 5; i++){

            printint(test[i]);

      }

      _exit(0);

      return 0;

}
```

testreverse.c

```
#include <types.h>
#include <kern/errno.h>
#include <kern/unistd.h>
#include <kern/limits.h>
#include <lib.h>
#include <clock.h>
#include <thread.h>
#include <syscall.h>
#include <uio.h>
#include <vfs.h>
#include <sfs.h>
#include <test.h>
#include "opt-synchprobs.h"
#include "opt-sfs.h"
#include "opt-net.h"

int main(){
    const char* str = "This is a test string for reverse string.
"

    reversestring(str, 51);

    _exit(0);

    return 0;

}
```