System Calls in Unix and Windows

Vivek Vishnumurthy

Purpose of this Lecture

- To familiarize you with using system calls – especially the ones to do with accessing and manipulating files.
- Will be of use later in the semester!

What are System Calls?

- The services provided by the kernel to application programs.
- This interface is needed because there are things (eg: I/O instructions) that only the kernel is allowed to perform.

Roadmap

- File manipulations:
  - `open, close, creat, rename`
  - `read, write, lseek`
- Directory manipulations:
  - `chdir, mkdir`
  - `link, unlink`
- stdin, stdout, stderr

Roadmap…

- Processes:
  - `getpid, execl, fork`
- Other:
  - `time`

But first, the Shell

- The shell provides a user interface to the system.
  - Often referred to as the “command-line interpreter”
  - But could be a GUI as well!
  - Egs: bash, csh, …

Issuing a System Call – Normal Procedure:

- In Unix: each system call is typically defined as a function in the standard C library.
- User process calls this function (in the normal C fashion)
- The function then invokes appropriate kernel service.
- Use "**`man callname`**" for info about the call.
- **"`man -s 2 callname`"** in Solaris.
- **"`man -S 2 callname`"** or just **"`man callname`"** in Linux.

7

Linux System Calls (about 300?)

_llseek	_newselec	_sysctl	accept	access	acct	adjtimex	afs_syscal	alarm	alloc_huge
arch_prctl	bdflush	bind	break	brk	cacheflush	capget	capset	chdir	chmod
chown	chroot	clone	close	connect	creat	create_mo	delete_mo	dup	dup2
epoll_crea	epoll_ctl	epoll_wait	execve	fchdir	fchmod	fchown	fcntl	fdatasync	flock
fork	free_huge	fstat	fstat64	fstatfs	fsync	ftruncate	futex	get_kernel	get_thread
getcontext	getdents	getdomain	getdtables	getegid	geteuid	getgid	getgroups	gethostid	gethostnan
getitimer	getpagesiz	getpeernar	getpgid	getpgrp	getpid	getpmsg	getppid	getpriority	getresgid
getresuid	getrlimit	getrusage	getsid	getsockna	getsockop	gettid	gettimeofd	getuid	gtty
idle	inb	inb_p	init_modul	ini	ini_p	insb	insl	insw	intro
inw	inw_p	io_cancel	io_destroy	io_geteven	io_setup	io_submit	ioctl	ioctl_list	ioperm
iopl	ipc	kill	killpg	lchown	link	listen	llseek	lock	lookup_dc
lseek	lstat	madvise	mincore	mkdir	mknod	mlock	mlock.2	mlockall	mlockall.2
mmap	mmap2	modify_ldt	mount	mprotect	mpx	mremap	msgctl	msgget	msgop
msgrcv	msgsnd	msync	munlock	munlockall	munmap	nanosleep	nfsservctl	nice	obsolete
oldfstat	oldstat	oldoldunar	oldstat	olduname	open	outb	outb_p	outl	outl_p
outsb	outsl	outsw	outw	outw_p	pause	pciconfig_	pciconfig_	pciconfig_	personality
pipe	pivot_root	poll	posix_fadv	prctl	pread	prof	pselect	ptrace	putpmsg
pwrite	query_mo	quotactl	read	readahead	readdir	readlink	readv	reboot	recv
recvfrom	recvmsg	rename	rmdir	sbrk	sched_get	sched_get	sched_get	sched_get	sched_get
sched_rr_	sched_set	sched_set	sched_set	sched_yie	security	select	select_tut	semctl	semget
semop	semtimed	send	sendfile	sendmsg	sendto	set_thread	setcontext	setdomain	setegid
seteuid	setfsgid	setfsuid	setgid	setgroups	sethostid	sethostnar	setitimer	setpgid	setpgrp
setpriority	setregid	setresgid	setresuid	setreuid	setrlimit	setsid	setsockop	settimeofd	setuid
setup	sgetmask	shmat	shmctl	shmdt	shmget	shmop	shutdown	sigaction	sigaltstack
sigblock	siggetmas	sigmask	signal	sigpause	sigpending	sigprocma	sigqueue	sigreturn	sigsetmask
sigsuspen	sigtimedw	sigvec	sigwaitinfo	socket	socketcall	socketpair	ssetmask	stat	statfs
statvfs	stime	stty	swapoff	swapon	symlink	sync	syscall	syscalls	sysctl
sysfs	sysinfo	syslog	time	times	tkill	truncate	umask	umount	umount2
uname	undocume	unimplemn	unlink	uselib	ustat	utime	utimes	vfork	vhangup
vm86	wait	wait3	wait4	waitpid	write	writev			

8

Issuing SysCall in Unix…

```
#include <lib-file.h>

   syscall-name();
```

Lib file that has function declaration

SysCall being called

9

In C#/Windows?

- "The Windows API"
  - (Comparatively) larger number of function calls.
- We will just concentrate on how to issue File I/O calls in C#.
- These calls are implemented in the **`System.IO`** namespace.
- In VS .NET : see help on members of **`System.IO.File`** and **`System.IO.FileStream`**

10

So, in C#…

```
using System.IO ;

   method-name();
```

Namespace that has method declaration

File Method being called

11

File Manipulation

- *File Descriptors* in Unix
- *File Streams* in C#
- These are basically identifiers of a file opened for data access.
  - Support reading from files.
  - Support writing to files.
  - Support seeking – changing position within the stream.

12

`open` (an exisiting file) in C/Unix

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
        ⋮

// open file 'test' for reading and
   writing
int fdtest = open("test", O_RDWR);
```

It's the option that sets file manipulation permissions –
Run "man 2 open" for more info

`open` (an exisiting file) in C#

```
using System.IO;
        ⋮

// open file 'test' for reading and writing
Stream fstream = File.Open("test",
                           System.IO.FileMode.Open,
                           System.IO.FileAccess.ReadWrite);
```

This is the option that sets file manipulation permissions –
See VS .NET help on "`FileAccess`" for more info

`create` (a new file)

C/Unix:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
        ⋮

//create newFile with read,
//write, execute permissions
int newfd = creat("newFile",
                  S_IRWXU);
```

C#:

```
using System.IO;
        ⋮

//create file newFile
Stream fstream =
   File.Create("newFile");
```

This is the option that sets required access permissions –
Run "man creat" for more info

`close` (an already opened descriptor or stream)

C/Unix:

```
#include <unistd.h>
        ⋮

//close opened descriptor
int closeResult =
   close(fdtest);
if(closeResult==0){
  printf("Closed
   successfully\n");
}else{
  printf("Close failed\n");
}
```

C#:

```
using System.IO;
        ⋮

//close opened stream
fstream.Close();
```


`read` (from an opened file)

```
#include <unistd.h>
        ⋮

int nBytes = 100;
char buf[nBytes]       ;
//read upto 'nBytes' bytes
// from file into buf
int nRead = read(fd, buf,
                 nBytes);
```

```
using System.IO;
        ⋮

byte[] bytes = new byte[100];
//read from file into 'bytes'
int nRead = fstream.Read(
                 bytes, 0,
                 bytes.Length);
```

The number of bytes actually read

`write` (to file)

C/Unix:

```
#include <unistd.h> //for write
#include <string.h> //for strlen
        ⋮

char buf[] = "Hello World!\n";
int nBytes = strlen(buf);
//write 'nBytes' bytes from buf
//to the file
int nWrote = write(fd,buf,nBytes);
```

Number of bytes written

Opened descr. on file

`write` (to file)

C#:

```
using System.IO;

String buf = "Hello World!\n";
byte[] bufBytes = new byte[buf.Length];
int nBytes = buf.Length;
for(int i=0; i<nBytes; i++)
{
   bufBytes[i] = (byte)buf[i];
}
//write 'nBytes' bytes from bufBytes to file
fstream.Write(bufBytes, 0, nBytes);
```

Offset from start

#Bytes to write

Opened stream on file

19

`lseek` in C/Unix

- Repositions read/write position in file.
- Useful when (say) you want to read bytes 20-29 from a file.

```
#include <sys/types.h>
#include <unistd.h>

//offset from start
int offset = 20;
//reposition read/write position
lseek(fd, offset, SEEK_SET);
char buf[10];
//read 10 bytes starting at offset
read(fd, buf, 10);
```

The offset is from the *start* of file. For other options: run "`man lseek`"

20

`Seek` in C#

```
using System.IO;

//offset from start
int offset = 20;
//reposition read/write position
fstream.Seek(offset, System.IO.SeekOrigin.Begin);

byte[] bufBytes = new byte[10];
//read 10 bytes starting at offset
fstream.Read(bufBytes, 0, bufBytes.Length);
```

The offset is from the *start* of file. For other options: see help on `SeekOrigin` in VS .NET

21

`rename` in C/Unix

- Used to change name/location of a file.

```
#include <stdio.h>

//rename file 'oldfile' to 'newfile'
int resRename = rename("oldfile",
   "newfile");
if(resRename==0){
   printf("rename successful\n");
}else{
   printf("rename failed\n");
}
```

22

`Move` in C#

- Used to change name/location of a file.

```
using System.IO;

//rename file 'oldfile' to 'newfile'
File.Move("oldfile", "newfile");
```

23

Directories: `chdir` (C/Unix)

- Change working directory.

```
#include <unistd.h>

char newDir[] = "/usr/u/vivi/courses/414";
//change working directory to newDir
chdir(newDir);
```

24

`mkdir` (C/Unix)

- Create new directory.

```
#include <unistd.h>

char newDir[] = "/usr/u/vivi/courses/414";
mkdir(newDir, S_IRWXU);
```

This is the option that sets access permissions – Run "`man creat`" for more info

25

`link` (C/Unix)

- Creates a new link (a *hard* link) to an existing file.

```
#include <unistd.h>
        ⋮
link("oldfile", "newname");
```

- Both **`oldfile`** and **`newname`** now refer to the same file.

26

`unlink` (C/Unix)

- Deletes a filename.

```
#include <unistd.h>
        ⋮
unlink("newname");
```

- If newname is the last link to the file, then file itself is deleted, else the link is deleted.

27

Processes: `getpid` (C/Unix)

- Every process has a unique process ID.
- **`getpid`**`()` returns the value of the ID of the calling process.

```
#include <unistd.h>
#include <sys/types.h>

//get process ID of the process
int pid = getpid();
```

- Used to generate unique temp file names.

28

`execl` (C/Unix)

- Replace current process image with new process image.

```
#include <unistd.h>
        ⋮
execl("/bin/ls", "~", NULL);
```

Arguments to the new process

The last argument has to be a null pointer

- If newname is the last link to the file, then file itself is deleted, else the link is deleted.

29

`fork` (C/Unix)

- Creates a replica child process differing only in pid and ppid(parent's pid).

```
#include <sys/types.h>
#include <unistd.h>
        ⋮
if(fork()==0)
   printf("I am the child\n");
else
   printf("I am the parent\n");
```

In parent, fork() returns child's pid. In child, fork() returns 0.

30

`time` (C/Unix)

- Get time in seconds - #seconds elapsed since 00:00:00 UTC, January 1, 1970.

```
#include <time.h>
        |
time_t elapsed = time();
printf("Time elapsed = %ld\n", elapsed);
```

31

`stdin/stdout/stderr` (Unix)

- Are streams for Standard Input, Output and Error, respectively.
- Each Unix program has these three streams opened for it.
- Eg: When **`printf`** or **`putchar`** is called, the output goes to **`stdout`**.
- Similarly, when **`getchar`** or **`scanf`** is called, the input is from **`stdin`**.
- **`stderr`** used to output error messages.

32

Using `stdin/stdout…` for simpler File I/O

- Say a program *prog* uses **`getchar`**.
- The command-line

  `$ prog < infile`

  causes *prog* to read its characters from **`infile`**, rather than the terminal.
- *prog*'s standard input stream now reads from infile.
- Examples taken from "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie.

33

`stdin/stdout…`

- Say a program *writer* writes out text, and program *reader* reads text. Then:

  `$ writer | reader`

  causes *reader* to read characters produced by *writer*.
- *reader*'s stdin is *writer*'s stdout.
- Examples taken from "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie.

34

Summary

- We looked at how to use system calls and do File I/O.
- Use the man and the help pages extensively!

35