SYSTEM CALLS

Module 1.5

Richard Newman

University of Florida

SYSTEM CALLS

- Regular Function/Procedure call mechanism
- System call mechanism
- Library procedures/API
- System call catalog
  - Process management
  - File management
  - Directory management
- A simple shell
- Window32 calls corresponding to Posix

PROCESS MEMORY LAYOUT

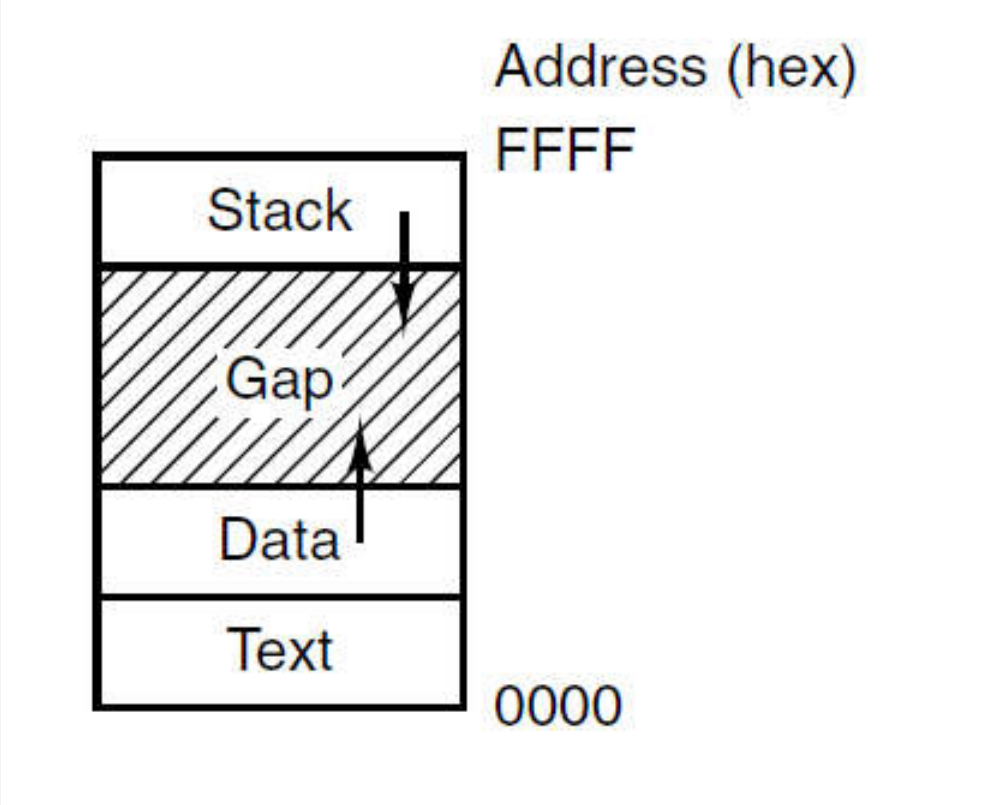


Figure 1-20. Processes have three segments: text, data, and stack

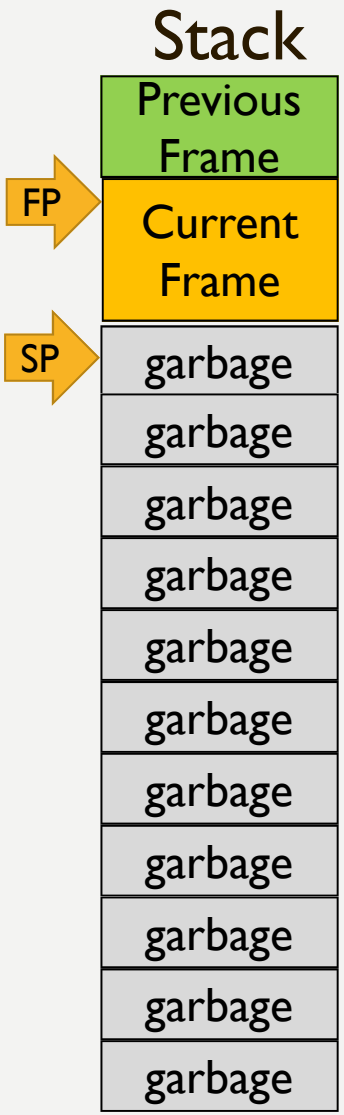
PROCEDURE CALLS

- Function is named code with
  - Local variables and parameters
  - Return to point whence called
- Jump to entry/return
  - Must store return address
- Parameters
  - Store on stack
  - Pass by value, by reference, by name
- Local vars
  - Stack

PROCEDURE CALLS

Frame pointer – points to bottom of previous frame

Stack pointer – points to byte below current frame

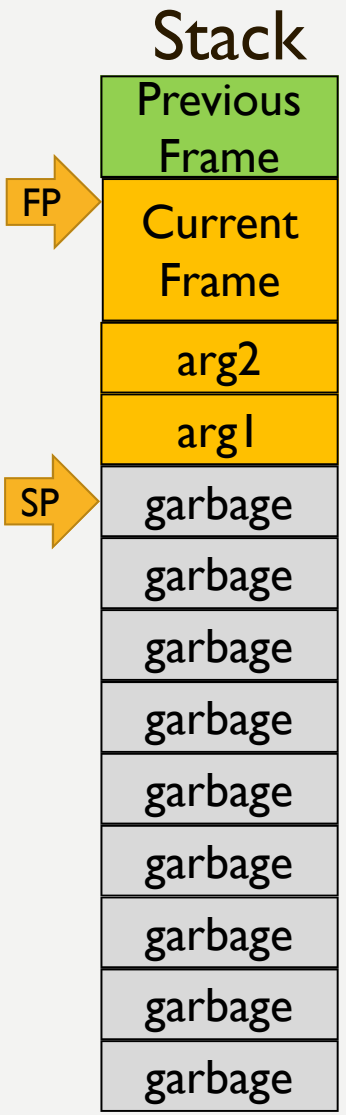


PROCEDURE CALLS

Frame pointer – points to bottom of previous frame

Stack pointer – points to byte below current frame

Parameters – usually passed by pushing onto current frame



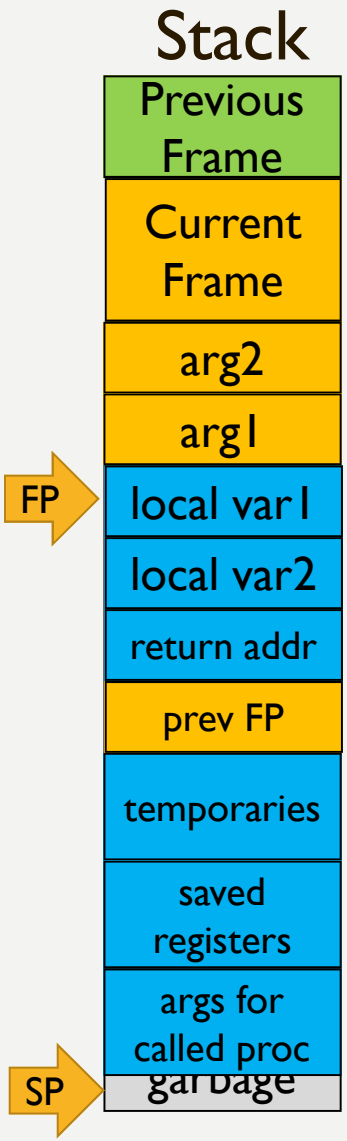
PROCEDURE CALLS

Frame pointer – points to bottom of previous frame

Stack pointer – points to byte below current frame

Parameters – usually passed by pushing onto current frame

Function call – new frame is pushed onto stack with local vars, return address, temporaries, saved registers

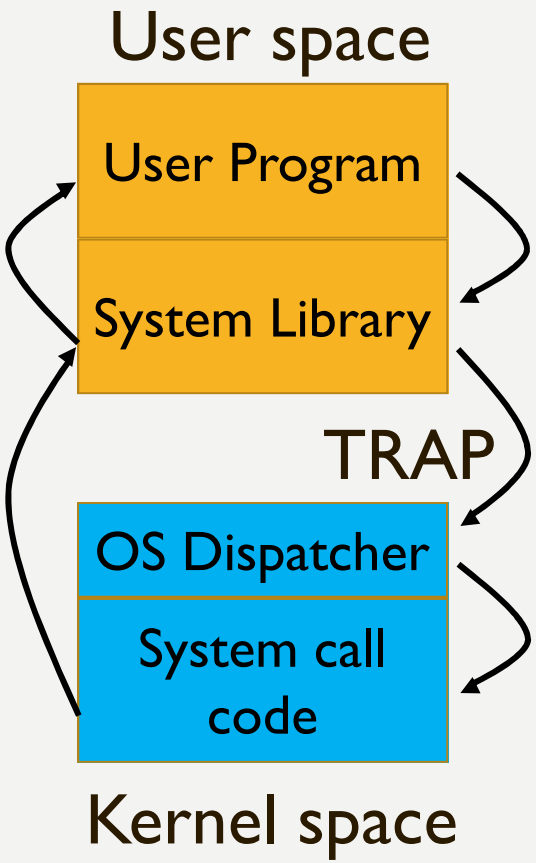


SYSTEM CALLS

- Also named code
  - Library procedure part of address space
  - If monolithic, then so is system call code
- Jump to entry/return
  - Always trap to kernel entry point
  - Must store return address, and *change state*
  - If microkernel, then store process state also!
- Parameters
  - Must provide system call number to kernel
  - Index into vector of functions

SYSTEM CALLS VS. LIBRARY

- User program calls system library
- Library is just archive of relocatable executable code – presents API
- Inside library routines, system call is actually made
- TRAP to kernel
- Kernel dispatcher reads syscall number, makes internal call to code that carries out the system call
- System returns to library
- Library routine returns to user program



SYSTEM CALLS

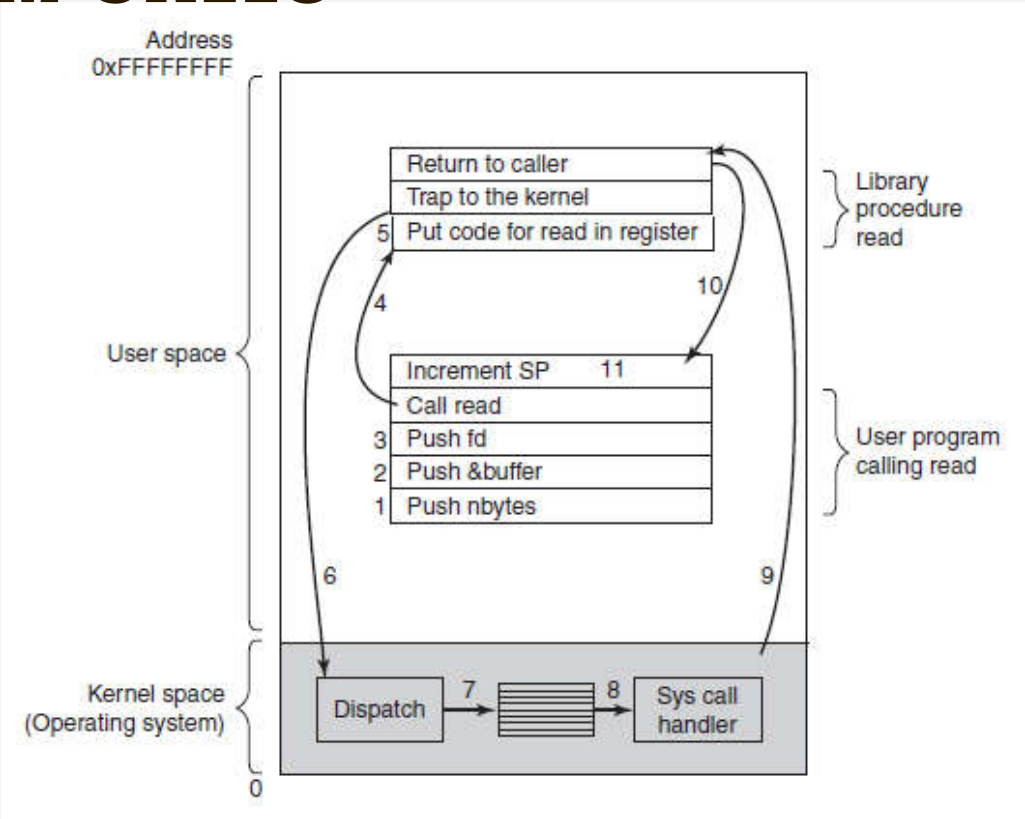


Figure 1-17. The 11 steps in making the system call *read(fd, buffer, nbytes).*

POSIX SYSTEM CALLS - PROCESSES

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 1-18. Some of the major POSIX system calls. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

POSIX SYSTEM CALLS - FILES

File management	
Call	**Description**
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 1-18. Some of the major POSIX system calls. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

POSIX SYSTEM CALLS - DIRECTORIES

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figure 1-18. Some of the major POSIX system calls. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

SYSTEM CALLS FOR DIRECTORY MANAGEMENT - LINK

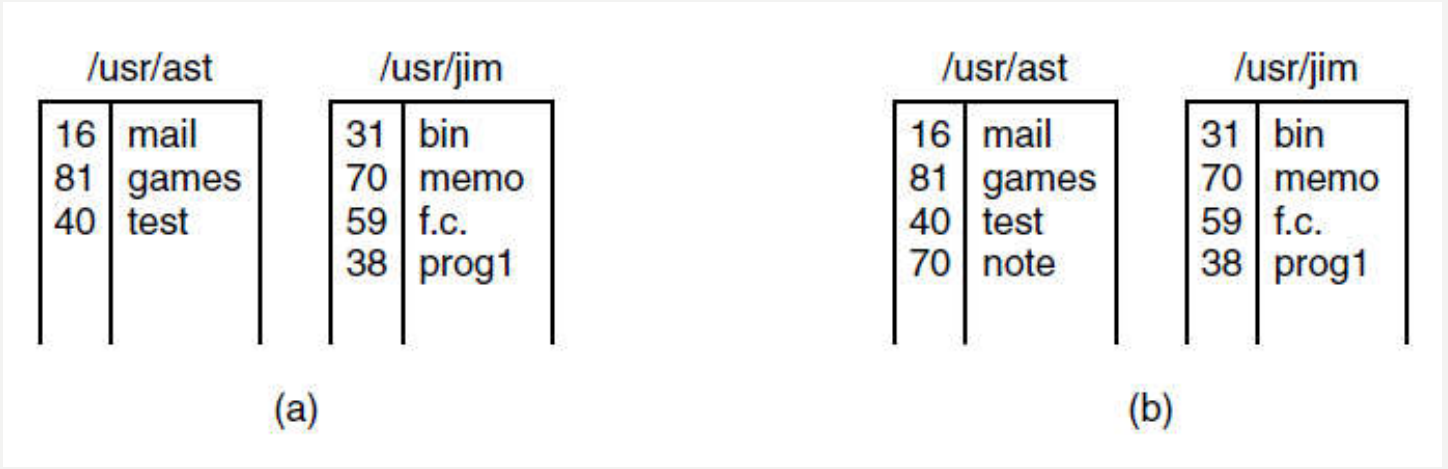
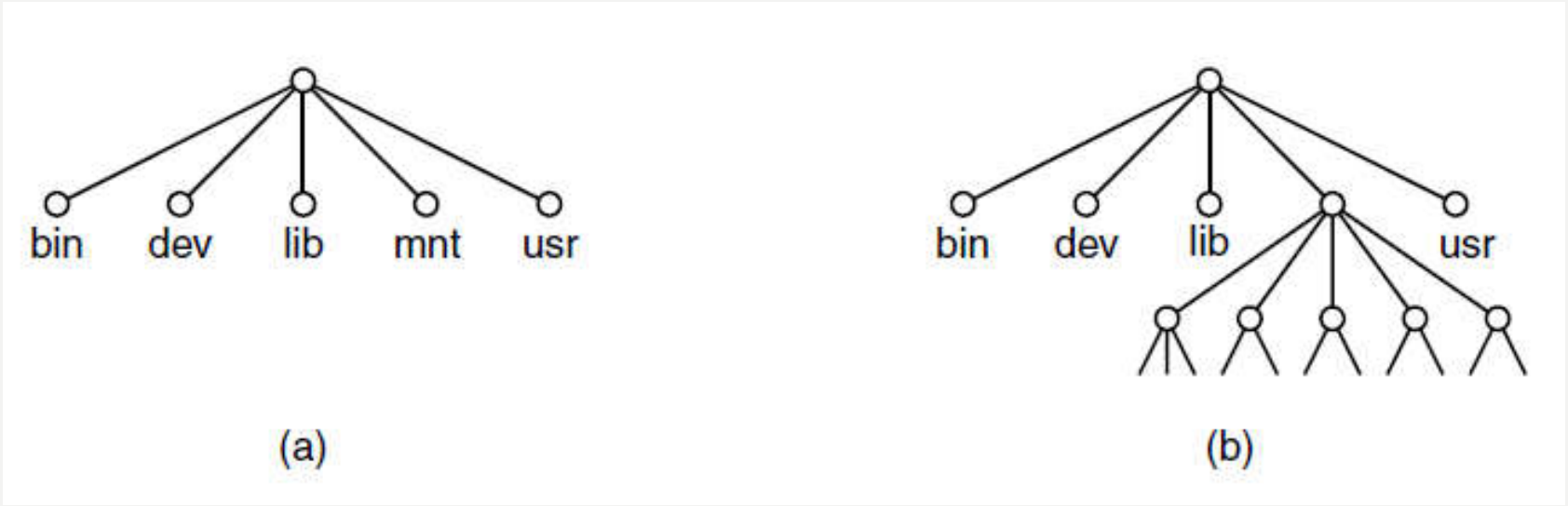


Figure 1-21. (a) Two directories before linking *usr/jim/memo* to *ast*'s directory. (b) The same directories after linking.

SYSTEM CALLS FOR DIRECTORY MANAGEMENT - MOUNT



Mounting FS on /mnt mount point

Figure 1-22. (a) File system before the mount.
(b) File system after the mount.

POSIX SYSTEM CALLS - MISC

Miscellaneous	
Call	**Description**
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figure 1-18. Some of the major POSIX system calls. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

SIMPLE SHELL CODE

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
      type_prompt( );                           /* display prompt on the screen */
      read_command(command, parameters);        /* read input from terminal */

      if (fork( ) != 0) {                       /* fork off child process */
           /* Parent code. */
           waitpid(–1, &status, 0);             /* wait for child to exit */
      } else {
           /* Child code. */
           execve(command, parameters, 0);      /* execute command */
      }
}
```

Figure 1-19. A stripped-down shell. Throughout this book, *TRUE* is assumed to be defined as 1.

THE WINDOWS WIN32 API (1)

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory

Figure 1-23. The Win32 API calls that roughly correspond to the UNIX calls of Fig. 1-18.

THE WINDOWS WIN32 API (2)

lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Figure 1-23. The Win32 API calls that roughly correspond to the UNIX calls of Fig. 1-18.