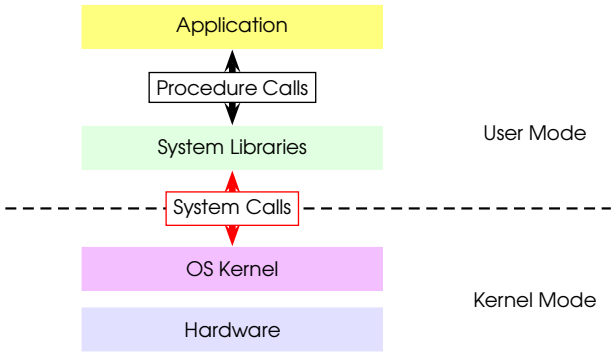
Slide 1

A Closer Look at System Calls

- User's view on system calls
- Implementation of System Calls

Slide 2



Slide 3

System Calls in Unix

- Process Management
  - `fork()`
  - `waitpid (pid, &statloc, options)`
  - `execve (name, argv, environp)`
  - `exit (status)`
  - `kill (pid, signal)`
- File Management
  - `open (file, modes)`
  - `close (fd)`
  - `read (fd, buffer, nbytes)`
  - `write (fd, buffer, nbytes)`
  - `lseek (fd, offset, whence)`
  - `stat (name, &buf)`

Slide 4

- File System Management
  - `mkdir (name, mode)`
  - `rmdir (name)`
  - `link` and `unlink`
  - `mount` and `unmount`

Slide 5

Win32 Application Programmer Interface

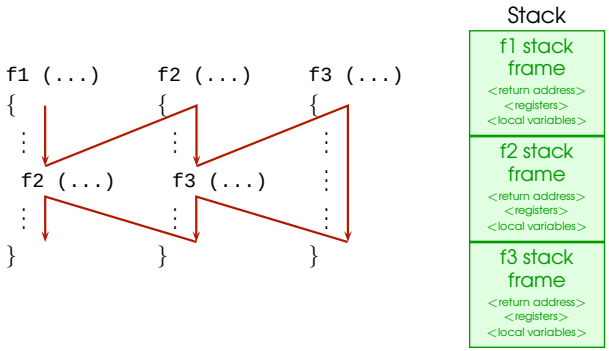
- Consists of hundreds of functions
- Many do not invoke system calls, carried out in user space
- Window management is part of the Win32 API, partially carried out in the kernel

Slide 6

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
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

Slide 7

What is the difference between a system call and a regular function call?



Slide 8

Procedure Calls

- For a procedure call, it is important that the caller and the callee agree on a certain protocol
- In theory, every compiler could use a different protocol
- Generally, compilers stick to the calling convention of the architecture

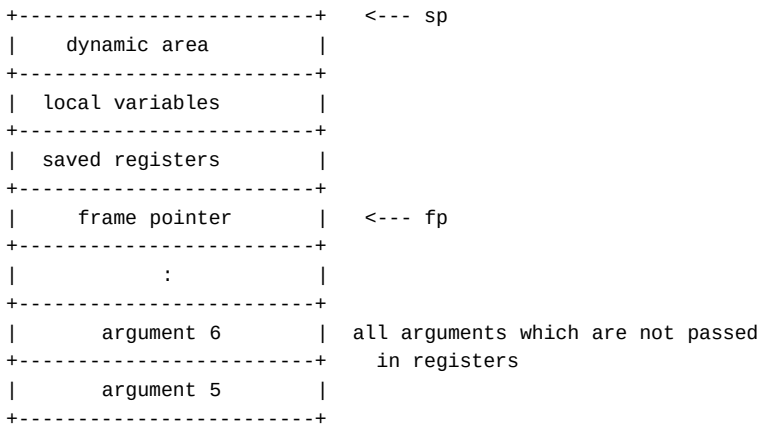
MIPS CALLING CONVENTION

Slide 9

Stack Layout:

- ➜ frame pointer is stored in register $30 ($fp)
- ➜ a stack frame consists of the memory on the stack between the frame pointer and the stack pointer.

Slide 10



Procedure Call — Caller:

Slide 11

① Copy first 4 arguments to registers $a0-$a3
② Push remaining arguments on the stack.
③ Save the caller-saved registers ($t0-$t9) if necessary
④ Execute jump and link (`jal`) instruction
- causes current pc to be saved (in $ra)

Procedure Call — Callee:

Slide 12

① Allocate space for stack frame (decrement frame size from stack pointer)
② Save the callee-saved registers in the frame:
- frame pointer ($fp)
- return address ($ra)
- arguments ($a0-$a3) if necessary
- registers $s0- $s7 if used by the callee

③ Update frame pointer(add stack frame size to $sp)

Slide 13

Return from call

① copy return value into register $v0
② restore callee-saved registers that were saved upon entry.
③ pop the stack frame (add frame size to $sp)
④ return by jumping to the address in register $ra.
⑤ restore caller saved register values

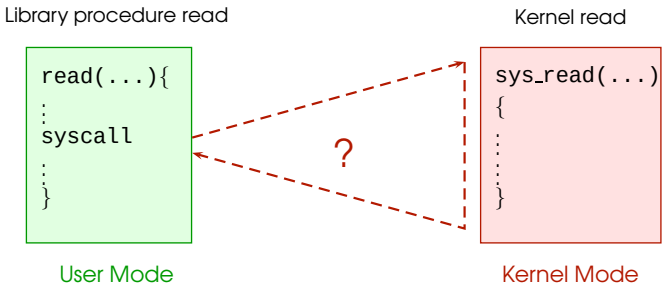
Slide 14

System Calls

Systems calls are different from procedure calls in two important aspects:

- ➜ Have to be executed in kernel mode
- ➜ For security reasons, they should not use the user stack, but separate kernel stack

Slide 15



- ➜ `syscall` to only way to switch to kernel mode
- ➜ causes an exception
- ➜ exception handler activated, not `sys_read`
- ➜ stack etc has to be set up "by hand"

Slide 16

Exception Handler

What does an exception handler do?

- ➜ saves current stack pointer
- ➜ switches to kernel stack
- ➜ save remainder of state (registers, etc)
- ➜ push trap frame on stack, so stack looks (almost) like a regular control stack
- ➜ find out what caused the exception?
  - syscall
- ➜ Which system call
  - check syscall number (set by syscall wrapper)
- ➜ call kernel function to handle system call
- ➜ return to wrapper