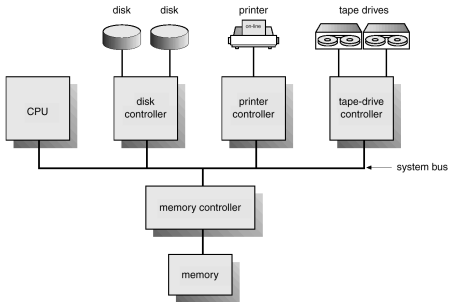
EE 469 Operating Systems Engineering

Hardware Support for Operating Systems

Spring 2001

David S. Ebert

Computer-System Architecture



Interrupts

A mechanism for achieving coordination between concurrently operating units of a computer system, and for responding to specific conditions within a processor.

A transfer of flow of control that is forced by the hardware.

Trap - a software generated interrupt caused either by an error, or by a user program request that an OS service be performed.

Basic I/O and Computer-System Operation

I/O devices and the CPU can execute concurrently.

Each device controller is in charge of a particular device type.

Each device controller has a local buffer and special purpose registers.

CPU moves data from/to main memory to/from local buffers

I/O is from the device to local buffer of controller.

***Device controller informs CPU that it has finished its operation by causing an* interrupt.**

Common Functions of Interrupts

Interrupts transfers control to the interrupt service routine generally, through the* interrupt vector*, which contains the addresses of all the service routines.

Interrupt architecture must save the address of the interrupted instruction.

Incoming interrupts are* disabled *(at this and lower priority levels) while the interrupt is being processed to prevent a* lost interrupt*.

Interrupts are enabled after servicing current interrupt

Modern operating systems are* interrupt *driven.

Interrupt Handling

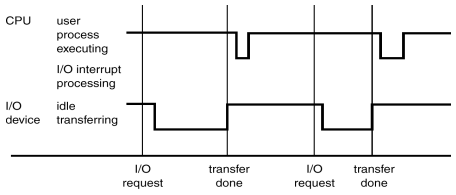
The operating system preserves the state of the CPU by storing registers and the program counter.

Determines which type of interrupt has occurred:

- *polling*
- vectored interrupt system

Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Time Line For a Single Process Doing Output



I/O Structure

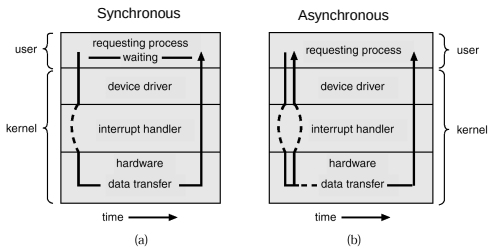
Synchronous I/O: After I/O starts, control returns to user program only upon I/O completion.

- wait instruction idles the CPU until the next interrupt
- wait loop (contention for memory access).
- At most one I/O request is outstanding at a time, no simultaneous I/O processing.

Asynchronous I/O: After I/O starts, control returns to user program without waiting for I/O completion.

- *System call* – request to the operating system to allow user to wait for I/O completion.
- *Device-status table* contains entry for each I/O device indicating its type, address, and state.
- Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Two I/O methods



Direct Memory Access (DMA) Structure

Used for high-speed I/O devices able to transmit information at close to memory speeds.

Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.

Only one interrupt is generated per block, rather than the one interrupt per byte.

Storage Structure

Main memory – only large storage media that the CPU can access directly.

Secondary storage – extension of main memory that provides large nonvolatile storage capacity.

Magnetic disks – rigid metal or glass platters covered with magnetic recording material

- Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
- The *disk controller* determines the logical interaction between the device and the computer.

Interrupt Driven O.S.

Modern OS's are interrupt driven

- Buffering and spooling required I/O interrupts
- Multiprogramming required artificial interrupts (traps) to switch from user to supervisor mode

At the lowest level an OS is just a bunch of interrupt service routings

- Each routine simply returns to whatever was executing before it was interrupted
  - *A use process*
  - *An OS process*
  - *Another interrupt routine*
- Else infinite wait loop

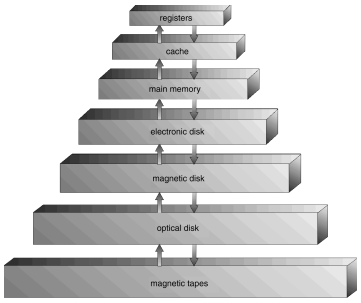
Storage Hierarchy

Storage systems organized in hierarchy.

- Speed
- cost
- volatility

Caching – ***copying information into faster storage system; main memory can be viewed as a*** **cache** ***for secondary storage.***

Storage-Device Hierarchy



Hardware Protection

Dual-Mode Operation

I/O Protection

Memory Protection

CPU Protection

Dual-Mode Operation

Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.

Provide hardware support to differentiate between at least two modes of operations.

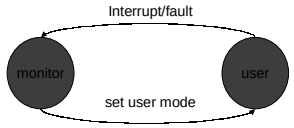
1. *User mode* – execution done on behalf of a user.
2. *Monitor mode* (also *supervisor mode*, *kernel mode* or *system mode*) – execution done on behalf of operating system.

Special system call instruction: generate an interrupt and switches to monitor mode (conversely, 1 to switch to user mode)

Dual-Mode Operation (Cont.)

Mode bit *added to computer hardware to indicate the current mode: monitor (0) or user (1).*

When an interrupt or fault occurs hardware switches to monitor mode.



Privileged instructions *can be issued only in monitor mode*.

Types of Protection

I/O Protection

Memory Protection

Cpu Protection

I/O Protection

All I/O instructions are privileged instructions.

Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

Memory Protection

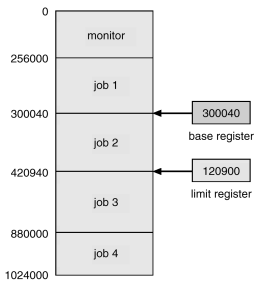
Must provide memory protection at least for the interrupt vector and the interrupt service routines.

In order to have memory protection, add two registers that determine the range of legal addresses a program may access:

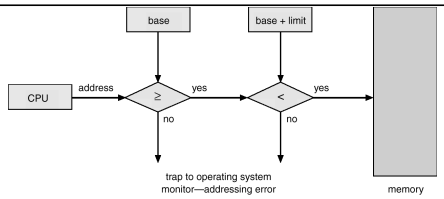
- **base register** – holds the smallest legal physical memory address.
- **Limit register** – contains the size of the range

Memory outside the defined range is protected.

A Base And A limit Register Define A Logical Address Space



Protection Hardware



When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.

The load instructions for the* base *and* limit *registers are privileged instructions.

CPU Protection

Timer ***– interrupts computer after specified period to ensure operating system maintains control.***

- Timer is decremented every clock tick.
- When timer reaches the value 0, an interrupt occurs.

Timer commonly used to implement time sharing.

Time also used to compute the current time.

Load-timer is a privileged instruction.

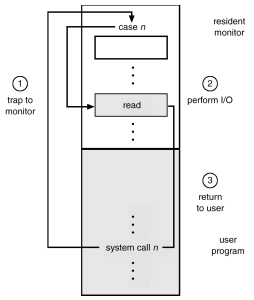
General-System Architecture

Given the I/O instructions are privileged, how does the user program perform I/O?

System call – the method used by a process to request action by the operating system.

- Usually takes the form of a trap to a specific location in the interrupt vector.
- Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to monitor mode.
- The monitor verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

Use of A System Call to Perform I/O



What Privileged Instructions are there in Unix?

Intro.2	getauid.2	readv.2
__sparc_utrap_install.2	getcontext.2	rename.2
_exit.2	getdents.2	resolvepath.2
_lwp_cond_broadcast.2	getegid.2	rmdir.2
_lwp_cond_signal.2	geteuid.2	sbrk.2
_lwp_cond_timedwait.2	getgid.2	semctl.2
_lwp_cond_wait.2	getgroups.2	semget.2
_lwp_continue.2	getitimer.2	semop.2
_lwp_create.2	getmsg.2	setaudit.2
_lwp_exit.2	getpgid.2	setauid.2
_lwp_getprivate.2	getpgrp.2	setcontext.2
_lwp_info.2	getpid.2	setegid.2
_lwp_kill.2	getpmsg.2	seteuid.2
_lwp_makecontext.2	getppid.2	setgid.2
_lwp_mutex_lock.2	getrlimit.2	setgroups.2
_lwp_mutex_trylock.2	getsid.2	setitimer.2
_lwp_mutex_unlock.2	getuid.2	setpgid.2
_lwp_self.2	intro.2	setpgrp.2
_lwp_sema_init.2	ioctl.2	setregid.2
_lwp_sema_post.2	kill.2	setreuid.2
_lwp_sema_trywait.2	lchown.2	setrlimit.2
_lwp_sema_wait.2	link.2	setsid.2
_lwp_setprivate.2	llseek.2	setuid.2
_lwp_sigredirect.2	lseek.2	shmat.2
_lwp_suspend.2	lstat.2	shmctl.2

More system calls

_lwp_wait.2	memcntl.2	shmdt.2
_signotifywait.2	mincore.2	shmget.2
access.2	mkdir.2	shmop.2
acct.2	mknod.2	sigaction.2
acl.2	mmap.2	sigaltstack.2
adjtime.2	mount.2	sigpending.2
alarm.2	mprotect.2	sigprocmask.2
audit.2	msgctl.2	sigsend.2
auditon.2	msgget.2	sigsendset.2
auditsvc.2	msgrcv.2	sigsuspend.2
booktitles.ent@	msgsnd.2	sigwait.2
brk.2	munmap.2	smancommon.ent@
chdir.2	nice.2	stat.2
chmod.2	ntp_adjtime.2	statvfs.2
chown.2	ntp_gettime.2	stime.2
chroot.2	open.2	swapctl.2
close.2	p_online.2	symlink.2
creat.2	pathconf.2	sync.2
dup.2	pause.2	sysfs.2
exec.2	pcsample.2	sysinfo.2

Even More

execl.2	*pipe.2*	*time.2*
execle.2	*poll.2*	*times.2*
execlp.2	*pread.2*	*uadmin.2*
execv.2	*priocntl.2*	*ulimit.2*
execve.2	*priocntlset.2*	*umask.2*
execvp.2	*processor_bind.2*	*umount.2*
exit.2	*processor_info.2*	*uname.2*
facl.2	*profil.2*	*unlink.2*
fchdir.2	*pset_assign.2*	*ustat.2*
fchmod.2	*pset_bind.2*	*utime.2*
fchown.2	*pset_create.2*	*utimes.2*
fchroot.2	*pset_destroy.2*	*vfork.2*
fcntl.2	*pset_info.2*	*vhangup.2*
fork.2	*ptrace.2*	*wait.2*
fork1.2	*putmsg.2*	*waitid.2*
fpathconf.2	*putpmsg.2*	*waitpid.2*
fstat.2	*pwrite.2*	*write.2*
fstatvfs.2	*read.2*	*writev.2*
getaudit.2	*readlink.2*	*yield.2*

From ls /usr/man/sman2