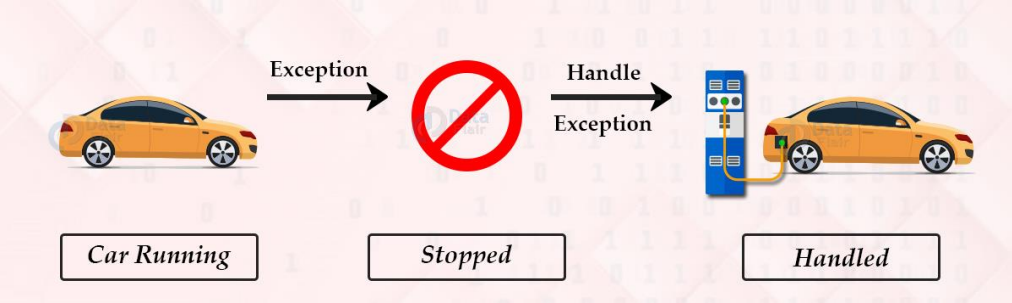
Exceptional Control Flow

COMPUTER SYSTEMS ORGANIZATION 2021

Control Flow

- Right from when you switch on your computer, the computer moves through a sequence of addresses a_k, a_k+1, a_k+2 and so on.
- Each *a_k* is the address of some corresponding instruction *Ik*. Each transition from *a_k* to *a_k*+1 is called a *control transfer*.
- A sequence of such control transfers is called the *flow of control*, or *control flow* of the processor.



Smooth Transition
- Branch, function call, code Sequence

Jittery Transitions
- Exception, fault, HDD failure

Exceptional Control Flow (ECF)

- Exceptional control flow occurs at all levels of a computer system.
- At the hardware level, events detected by the hardware trigger abrupt control transfers to exception handlers.
- At the operating systems level, the kernel transfers control from one user process to another via context switches.
- At the application level, a process can send a signal to another process that abruptly transfers control to a signal handler in the recipient.
- An individual program can react to errors by sidestepping the usual stack discipline and making nonlocal jumps to arbitrary locations in other functions.

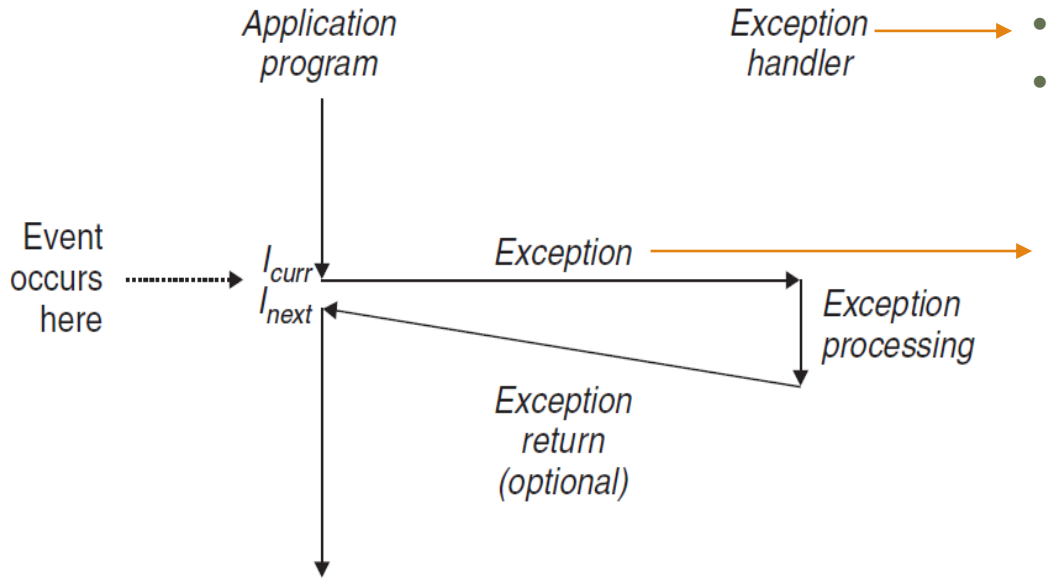
Why ECF

- ECF is the basic mechanism that operating systems use to implement I/O, processes, and virtual memory.
- Applications request services from the operating system by using a form of ECF known as a trap or system call.
- The operating system provides application programs with powerful ECF mechanisms for creating new processes, waiting for processes to terminate, notifying other processes of exceptional events in the system, and detecting and responding to these events.
- Languages such as C++ and Java provide software exception mechanisms via try, catch, and throw statements.

Exception

- Exceptions are not Exceptions they are everywhere.
- An exception is an abrupt change in the control flow in response to some change in the processor's state.

Figure 8.1
Anatomy of an exception.
A change in the processor's state (event) triggers an abrupt control transfer (an exception) from the application program to an exception handler. After it finishes processing, the handler either returns control to the interrupted program or aborts.



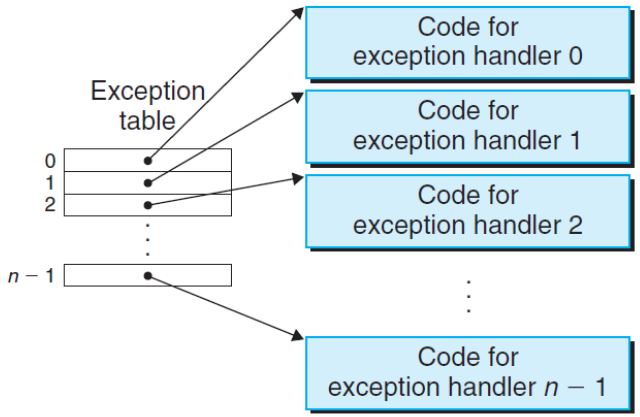
- Return the present Instruction
- GOTO next instruction
- ABORT

- a virtual memory page fault
- an arithmetic overflow occurs
- instruction attempts a divide by zero.
- system timer goes off or an I/O request completes.

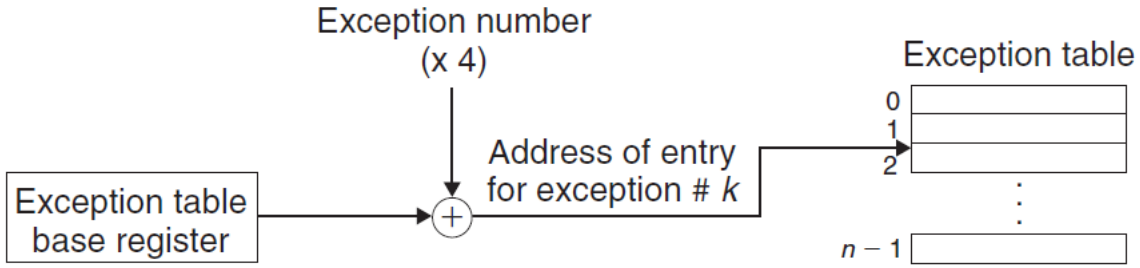
Handling the Exception

- Each type of possible exception in a system is assigned a unique nonnegative integer exception number.

Figure 8.2
Exception table. The exception table is a jump table where entry k contains the address of the handler code for exception k.

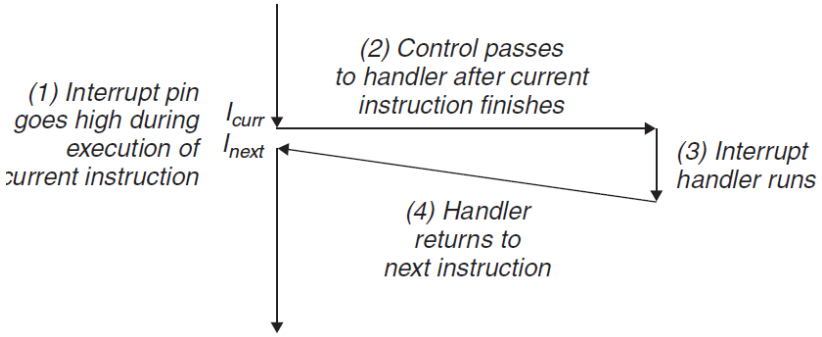


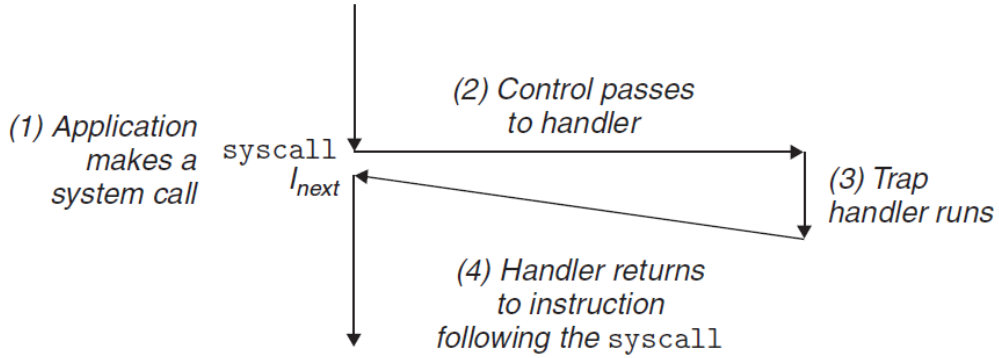
An exception is akin to a procedure call, but with some important differences.



Classes of Exception

- **Interrupts** - Interrupts occur asynchronously as a result of signals from I/O devices that are external to the processor. Example – Keyboard and mouse input.

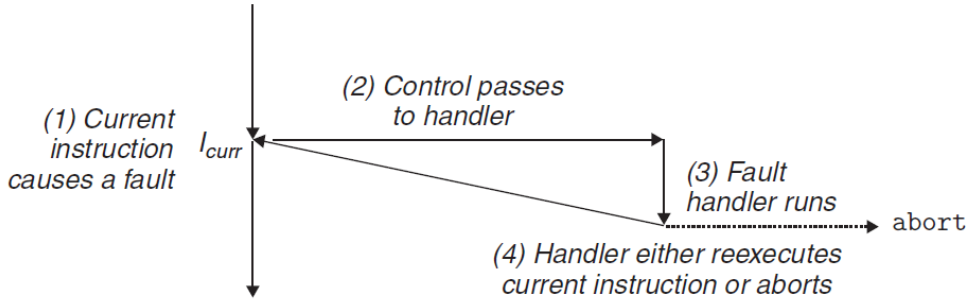




- **Traps and System Calls** - Traps are intentional exceptions that occur as a result of executing an instruction. Traps are used to implement system calls.

Classes of Exception

- **Faults** - Faults result from error conditions that a handler might be able to correct. A classic example of a fault is the page fault exception, which occurs when an instruction references a virtual address whose corresponding physical page is not resident in memory and must therefore be retrieved from disk.



- **Aborts -** Aborts result from unrecoverable fatal errors, typically hardware errors such as parity errors that occur when DRAM or SRAM bits are corrupted. Abort handlers never return control to the application program.

Practical Example

Exception number	Description	Exception class
0	Divide error	Fault
13	General protection fault	Fault
14	Page fault	Fault
18	Machine check	Abort
32–127	OS-defined exceptions	Interrupt or trap
128 (0x80)	System call	Trap
129–255	OS-defined exceptions	Interrupt or trap

Figure 8.9 **Examples of exceptions in IA32 systems.**

```
First, call write(1, "hello, world\n", 13)
 movl $4, %eax          System call number 4
 movl $1, %ebx          stdout has descriptor 1
 movl $string, %ecx     Hello world string
 movl $len, %edx        String length
 int $0x80              System call code
```

Number	Name	Description	Number	Name	Description
1	exit	Terminate process	27	alarm	Set signal delivery alarm clock
2	fork	Create new process	29	pause	Suspend process until signal arrives
3	read	Read file	37	kill	Send signal to another process
4	write	Write file	48	signal	Install signal handler
5	open	Open file	63	dup2	Copy file descriptor
6	close	Close file	64	getppid	Get parent's process ID
7	waitpid	Wait for child to terminate	65	getpgrp	Get process group
11	execve	Load and run program	67	sigaction	Install portable signal handler
19	lseek	Go to file offset	90	mmap	Map memory page to file
20	getpid	Get process ID	106	stat	Get information about file

Figure 8.10 **Examples of popular system calls in Linux/IA32 systems.** Linux provides hundreds of system calls. Source: `/usr/include/sys/syscall.h`.