Exceptional Control Flow

Exceptions

Exception - an abrupt change in the control flow in response to some change in the processor's state.

Event

Appln program

\[ \text{Exception handler} \]

\[ \text{Exception} \]

\[ \text{Exception processing} \]

\[ \text{Exception return (optional)} \]

Examples for events

1. VM page fault
2. Arithmetic overflow
3. Divide by zero
4. System timer goes off
5. I/O request completes

\[ \text{unrelated to the exec. of the curr instruction} \]
When the processor detects that the event has occurred:

1) It makes an **indirect procedure call** (the exception).
2) through a **jump table** called an exception table.
3) to an **OS subroutine** (the exception handler) that is specifically designed to process this particular kind of event.

After the exception handler finishes processing:

1. Current instruction (ICurr)
2. Next **in** (INext)
3. Handler can abort the interrupted program.

---

Exception Table

![Diagram of exception table]

**At system boot time,** the OS allocates and initializes a jump table (exception table).
Exception Table  Base Register

ETBR

Address of exception #4

Exception number

What happens when an exception occurs?

CPU

P

user mode

Exception Handler

Kernel mode

The CPU has context switched from running the user program P to running the exception handler.

CONTEXT SWITCH
User Stack vs Kernel Stack

A single process' address space.

Address space reserved by kernel

Exceptions

- Interrupts
- Faults
- Traps

Aborts
**Interrupts**

Hardware interrupts are "asynchronous" w.r.t. the current process.

- **Interrupt**
  - **Interrupt Request (IRQ)**
  - Device \rightarrow \text{processor}
  - **Processor halts current process**
  - **Processor saves process state**
  - **Processor executes interrupt handler**
  - **Processor resumes process exe**.
Why interrupts?

Possible Solution - Polling!

CPU periodically checks each device to see if it needs service.

"Polling is like picking up your phone every few seconds to see if you have a call."
Alternative: Interrupts
Each device is given a wire (interrupt line) that it can use to signal the processor.

"Interrupts are like waiting for the phone to ring!"

Traps and System Calls

Traps - intentional exceptions that occur as a result of executing an instruction.

1. App/ln makes sys call
2. Control passes to handler
3. Trap
4. Handler runs
5. Handler returns to instr. following the system call.
System Call

Application programs
\[\text{syscall}\]
Operating System

Hardware

1. open
2. read
3. write
4. close

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>exit</td>
</tr>
<tr>
<td>2</td>
<td>fork</td>
</tr>
<tr>
<td>3</td>
<td>read</td>
</tr>
<tr>
<td>4</td>
<td>write</td>
</tr>
<tr>
<td>5</td>
<td>open</td>
</tr>
<tr>
<td>6</td>
<td>close</td>
</tr>
</tbody>
</table>
Linux/IA 32 System Calls

\[ \text{int } n \]

\[ \text{interrupt} \rightarrow \text{exception number.} \]

\[ h = 0x80 (128_{10}), \text{ for system calls.} \]

```c
int main() {
    write(1, "hello, world\n", 13);
    exit(0);
}
```

Syscall # → stored in %eax.
Arguments → stored only in registers and not on stack.

%ebx, %ecx, %edx, %esi, %edi,

\[ \downarrow \quad \ldots \quad \%ebp \]

arg1

%esp will not be used, arg6.
**Faults**

Potentially recoverable error.

1. Cur instr causes a fault
2. Control passes to fault handler
3. Fault handler runs
4. Handler either reexecutes cur instr or aborts.

**Equation:** page fault exception, segmentation fault.

<table>
<thead>
<tr>
<th>Exception #</th>
<th>Description</th>
<th>Exception class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Divide error</td>
<td>Fault</td>
</tr>
<tr>
<td>13</td>
<td>General protection fault</td>
<td>&quot;</td>
</tr>
<tr>
<td>14</td>
<td>Page fault</td>
<td>II</td>
</tr>
<tr>
<td>18</td>
<td>Machine check</td>
<td>Abort</td>
</tr>
<tr>
<td>128 (0x80)</td>
<td>System Call</td>
<td>Trap</td>
</tr>
<tr>
<td>Other # (max 255)</td>
<td>OS-defined exceptions</td>
<td>Interrupt or Trap</td>
</tr>
</tbody>
</table>
.section .data
string:
  .ascii "hello, world\n"
string_end:
  .equ len, string_end - string

.section .text
.globl main
main:

# 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

# Next, call exit(0)
  movl $1, %eax        # System call number 0
  movl $0, %ebx        # Argument is 0
  int $0x80             # System call code