

Exceptions

CSE 361S

Normal Control Flow

Normal execution sequence is:

- next instruction immediately follows current instruction, or
- next instruction is result of jump, branch, call, or return instruction

Abnormal Control Flow

Alternative execution sequence can be triggered by:

- external event (e.g., interrupt)
- internal event (e.g., page fault, divide by zero)
- explicit request (e.g., ask for file I/O)

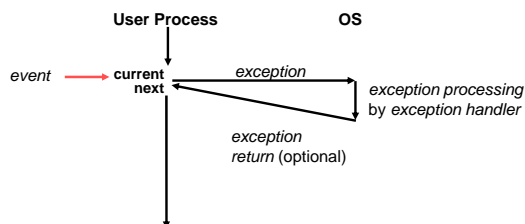
Text calls all of these “exceptions”

Some call only 2 and 3 exceptions, and refer to 1 as distinct class of things called “interrupts”

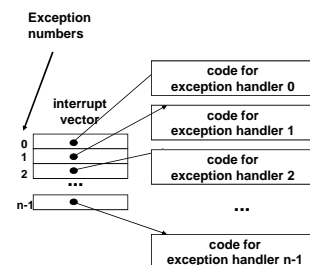
Exception Classes

trigger	name	timing	return
1	interrupt	asynchronous	next instruction
2	fault	synchronous	current instruction (?)
3	abort	synchronous	never
4	trap	synchronous	next instruction

OS Exception Handling



What code to execute as the result of an exception is typically managed by an explicit table (whose entries are often called interrupt vectors)



Trigger 1: Interrupts

Occur asynchronously wrt currently executing code.
E.g., timer, input signal from peripheral such as network packet or disk sector arrival.

- Action:
 - complete current instruction
 - branch to vectored Interrupt Service Routine (ISR)
 - return to next instruction
- When writing ISRs:
 - keep it short
 - handle immediate requirements, queue work for later
 - serious potential for priority inversion

Trigger 2: Faults

Occur synchronously with current instruction.

E.g., page fault, protection fault, floating point exception, TLB miss.

- Action:
 - branch to vectored exception handler
 - handler decides whether to:
 - abort, or
 - return and restart current instruction

Trigger 3: Aborts

Unrecoverable hard error.

E.g., corrupted memory, processor fault.

- Action:
 - terminate program
 - question – is termination always best?

Trigger 4: Traps

Explicitly requested exception.

E.g., transition to supervisor mode to access restricted item, request file I/O.

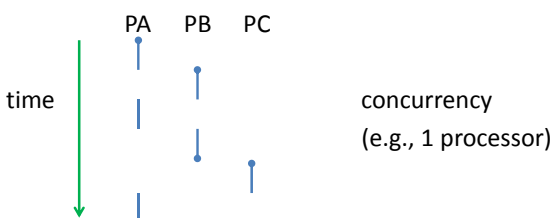
- Action:
 - branch to vectored exception handler
 - OS provides requested service at its convenience
 - return to next instruction

For all triggers, resumption of execution is whenever scheduled by the OS.

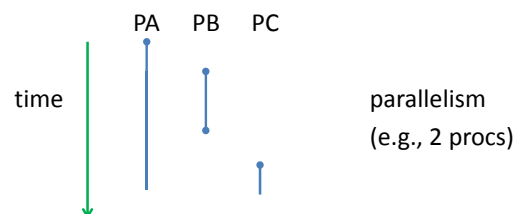
Processes

A process is an instance of a running program

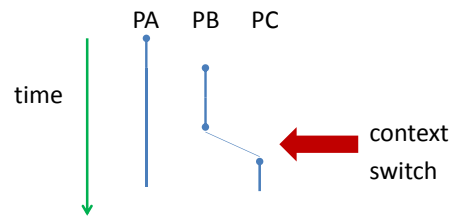
- Logical flow control (determined by code)
- Private virtual address space



Alternative Timeline



Context Switch



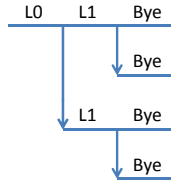
Creating New Processes

```
int fork(void);

if (fork() != 0) {
    printf("Parent process.\n");
}
else {
    printf("Child process.\n");
}
```

Example Usage

```
void foo() {
    printf("L0");
    fork();
    printf("L1");
    fork();
    printf("Bye");
}
```



Different Process

- Use `execve()` to load and run a new program in the context of the current process.

```
int execve(char *filename,
           char *argv[],
           char *envp[]);
```

Process Termination

- Return from `main()`, or
- Call `exit()`

```
void exit(int status)
```
- Parent process learns status via `wait()`