Context of a Process

- Context of a process
  - Contents of user address space **(user-level context)**
  - Contents of hardware registers **(register context)**
  - kernel data structure that relate to the process **(system-level context)**
- User-level context
  - **text, data, user stack, shared memory**
  - register context
  - **Program Counter (PC)** – specifies address of the next instruction the CPU will execute.
  - **Processor Status register (PS)** – Specifies h/w status of machine.
  - **User Stack Pointer (SP)** – contains current address of next entry in kernel or user stack.
  - **General purpose registers** – contain data generated by the process during execution.

Context of a Process

- System-level context
  - Static Part-
    - Process table entry,
    - U area of a Process,
    - Pregion entries , region tables , page tables
  - Dynamic Part-
    - Kernel Stack - Stack Frames,
    - Set of layers.

Context of a Process

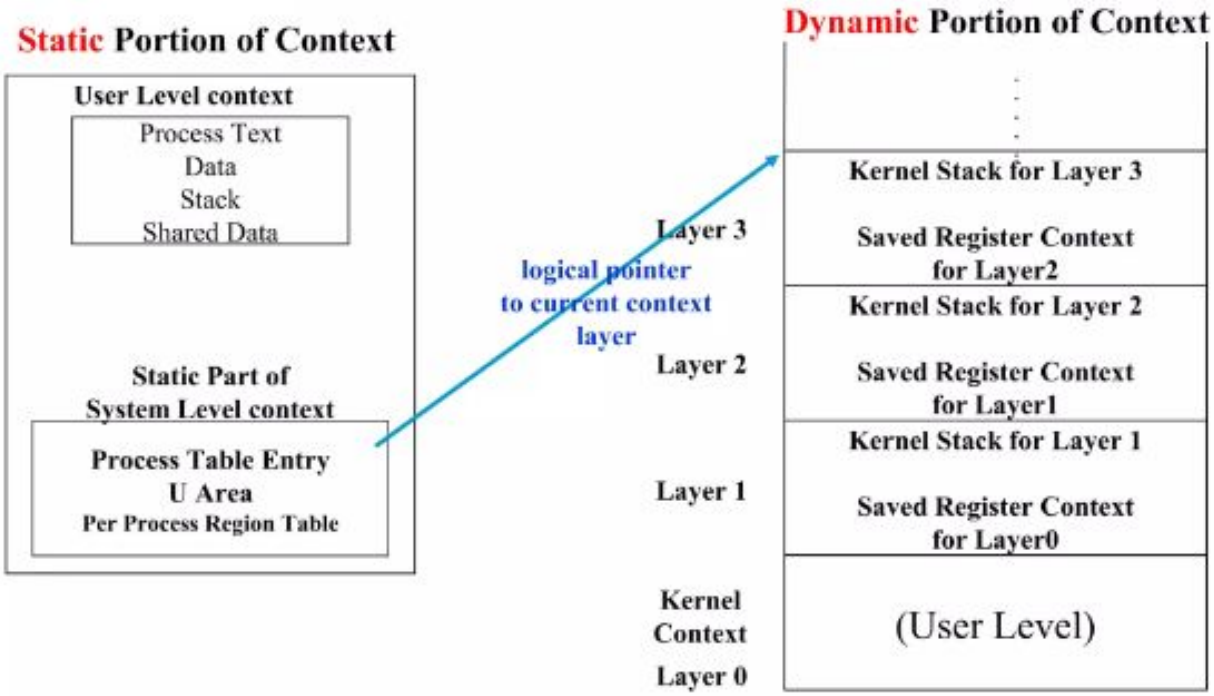


Fig: Components of the Context of a Process>

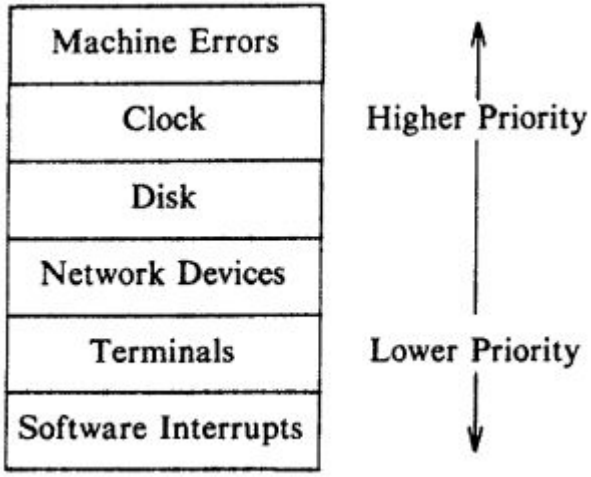
Saving the Context of a Process

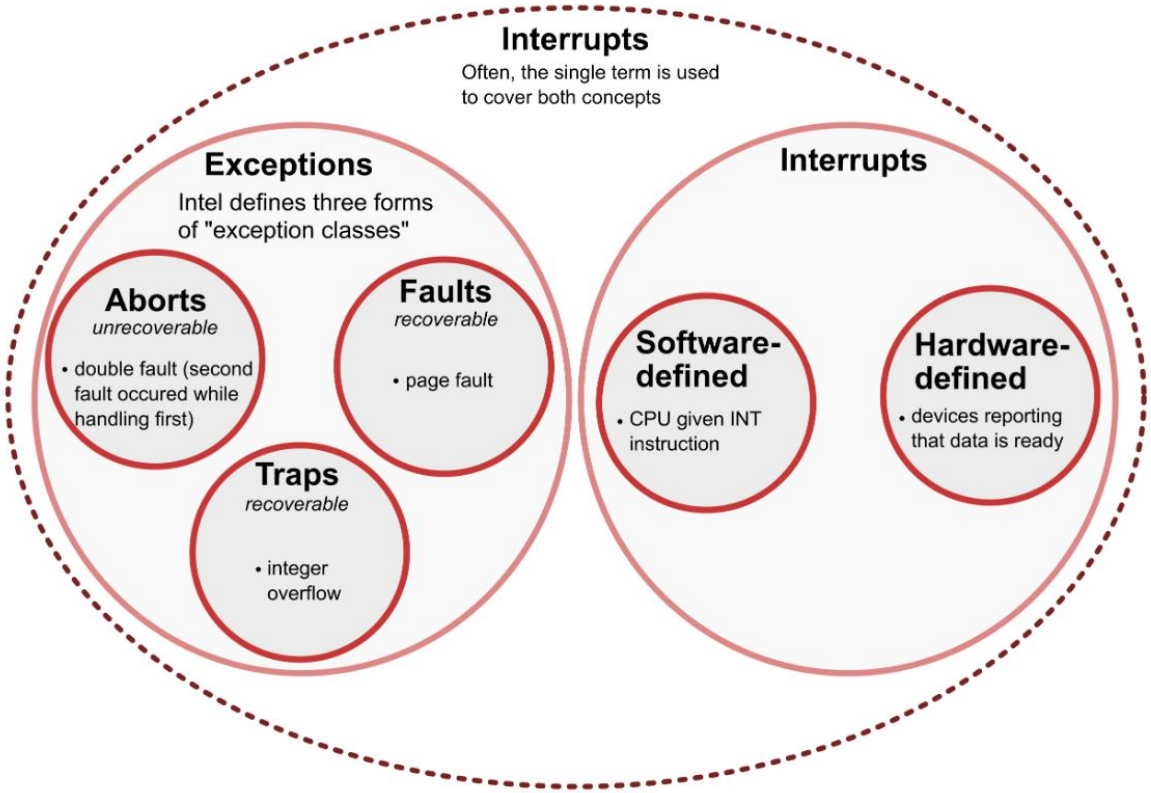
- Kernel saves the context of a process
  - Whenever it pushes a new system context layer.

- Interrupts and Exception
- System call Interface
- Context switch.

Interrupts and Exception

- System is responsible for handling interrupts
  - result from hardware (clock, peripheral)
  - programmed interrupt (software interrupt)
  - exceptions (page faults)
- Integrity of kernel data structure





Interrupts and Exception

- Sequence of interrupt operation
  1. Saves the current register context of the executing process and creates(push) a new context layer.
  2. Determine interrupt source and cause.
     - find interrupt vector.
  1. Kernel invokes interrupt handler.
  2. Interrupt handler completes it work and return.
     - Restores previous context layer.

Interrupts and Exception

```
algorithm inthand          /* handle interrupts */
input:  none
output: none
{
        save (push) current context layer;
        determine interrupt source;
        find interrupt vector;
        call interrupt handler;
        restore (pop) previous context layer;
}
```

Figure 6.10. Algorithm for Handling Interrupts

System Call Interface

- Usual calling sequence
  - cannot change the mode of a process form user to kernel
- Library : have the system call name
  - User program system call
- OS trap
- Same as Interrupt Handler

Context Switch

- Kernel permits a context switch
  - When a process puts itself to sleep
  - When it exits
  - When it returns from a system call to user mode but is not the most eligible process to run
  - When it returns to user mode after the kernel completes handling an interrupt but is not the most eligible process to run.

Context Switch

- Ensure integrity and consistency.
- Procedure for a context switch.
  - similar to handling interrupts and system call
- Steps for a Context Switch
  1. Decide whether to do a context switch, and whether a context switch is permissible now.
  2. Save the context of the "old" process.
  3. Find the "best" process to schedule for execution, using the process scheduling algorithm.
  4. Restore its context.

Manipulation of Process Address Space

- System calls Manipulate the virtual address space of a process.
- The region table entry contains the information necessary to describe a region.

Manipulation of Process Address Space

- Region Table contains the following entries:
  - A **pointer to the Inode** of the file whose contents were originally loaded into the region.
  - **Region type** (text, shared memory, private data or stack).
  - **Size of the region.**
  - The **location of the region in physical memory**.
  - The **status of a region**, which may be a combination of
    - Locked
    - In demand
    - In the process of being loaded into memory
    - Valid, loaded into memory
  - The **reference count**, giving the number of processes that reference the region.

Manipulation of Process Address Space

- **Operations on Region:**
  - Locking and unlocking a Region.
  - Allocating a Region.
  - Attaching a Region to a Process.
  - Changing the Size of a Region.
  - Loading a Region.
  - Freeing a Region.
  - Detaching a region from a Process.
  - Duplicating a Region.