Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Overview:

Processes.

PCB (Process Context Block).

Interrupts.

Context switching.

DMA (Direct Memory Access).
Processes, process states, and PCBs
Processes and Programs

Definition

**Process:** Is an instance of a computer program that is being executed.

- A process consists of:
  - **An address space:** containing the program code and the data (static, heap, and stack).
  - **The CPU state:** The values of CPU registers including PC and SP.
  - **A set of OS resources:** open files, network connections, ...
  - **A lifetime:** created-executed-[interrupted-resumed]-terminated.

- A program is a passive entity whereas a process is its active form.

- Two types of processes: system processes that execute system codes, and user processes that execute user codes (but treated in the same way).

*Overall:* A process is all what you need to run or resume a program.
Processes and Programs

- System processes are executed in kernel mode (master mode) whereas user processes are executed in user mode (slave mode).

- Various terms are used: process, job, task.

- During its lifetime, a process is identified by a unique [temporary] number called PID (Process IDentifier) (e.g., in Linux `ps -aux` or `$ top`).
Process states (viz., Process State Graph)

During its lifetime, a process transits from one state to another as follows:

![Process State Graph]

Figure 1: Process State Graph
Process states (viz., Process State Graph)

The different states are defined as follows:

- **New**: Just been created (e.g., in C language `fork();` is used).
- **Ready**: Having all needed resources and waiting (ready queue) for CPU to be allocated.
- **Running**: Being executed on the CPU.
- **Blocked**: Waiting (blocked queue) for another process to terminate (e.g., I/O routines) or an event to happen (e.g., signal reception).
- **Suspended**: Swapped out of RAM for some reason:
  - Swapped out of RAM to free some space for other process.
  - Swapped out of RAM when debugging the main program.
  - Intentionally suspended by superuser ([8am-6pm]) to save CPU cycles.
  - Swapped out of RAM after being longly waiting for an event to happen.
  - Swapped out by the parent process.
- **Terminated**: The process has finished its task and/or have been killed (e.g., in Linux `$ kill 13000` is used to terminate the process which PID is 13000).
Every process is represented by a data structure called PCB (Process Control Block) that stores information about the process last state:

<table>
<thead>
<tr>
<th>PID</th>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Enough information to resume execution and track process.
Process Creation and Termination (UNIX Style)

**Creation:** A process can create other processes (children).

- In UNIX systems, a process executes the `fork()` system call.
- The parent process can continue its execution or wait (`wait()`).
- A parent process can pass input data to its children.
- The address space can be, either a duplication of the parent space (same data and code), or a newly loaded program (`execvp()`).
- Resources are inherited (e.g., open files, privileges, ...).

**Termination:**

- A process executes the `exit()` system call.
- Running `exit()` returns an integer to the parent.
- A parent process can terminate a child `kill()`.
Concurrent Processes

Processes are said to be concurrent when:

- Multiple programs are loaded in the main memory RAM.
- Their execution is happening in parallel or in fake-parallel.
- Their time durations or executions are overlapping or interleaving.
- Processes execution order can be visualized by the PPG (Process Precedence Graph).
- An execution can be sequential (serial) or parallel.

![Processes Precedence Graph and pseudocode]

Figure 2: Processes Precedence Graph (Left) and its pseudocode (Right)
**Interprocess Communication**

Processes can cooperate with each other for several reasons:

- **Information sharing**: Collaborating on the same file.
- **Computational speedup**: Parallel execution of different parts of a program (multiprocessors systems only).
- **Modularity**: An operating system can be constructed in a modular fashion.
- **Convenience**: An individual user may run different processes.

Two fundamental mechanisms:

1. **Shared memory**: Need a shared memory region.
2. **Message passing**: Need a physical/logical communication link (local: pipes, remote: RPC and sockets).
Interrupts & Interrupt handlers
**Definition**

**Interrupt**: is an event that alters the sequence of instructions executed by the CPU. A dedicated program called ISR (Interrupt Service Routine) or interrupt handler is executed as a consequence.

Each interrupt has its own interrupt handler (written in assembly language).

**Definition**

**Interrupt handler**: a.k.a, ISR (Interrupt Service Routine), is a low level program (assembly) that is executed in kernel mode to service the cause of the interrupt.

Interrupt handlers are located by an IVT (Interrupt Vector Table) stored in the memory: e.g., IDT (Interrupt Descriptor Table) in x86 architectures generally stored in 0x0000-0x03FF.

The first handler is by convention the handler for the Timer.
Types of Interrupts

**Software-based Interrupts**: When the system is interrupted following the execution of an instruction in a program code, the interrupt is said to be software-based. There are two types:

1. **Exceptions**: An exception happens when the current instruction perform an illegal action such as: division by 0, arithmetic overflow, buffer overflow, page fault, access protected memory space, or attempt to execute a privileged instruction.

2. **System calls**: A.k.a., traps. User program are executed in user mode (slave mode). In this mode, programs do not have direct access to peripheral. They have to issue a **system call** which causes an interrupt.

**Hardware-based Interrupts**: These interrupts are trigged by a hardware unit such as the timer, keyboard, mouse, power button, ..., (device controllers or DMA) to get the attention from the operating system. They can be maskable (MI) or non-maskable (NMI).
Part 2: Process Management

Interrupts

Types of Interrupts

Other types of interrupts:

- **IPI (Inter-processor interrupt)**: Used between processors to communicate in a multiprocessor system e.g., one processor core requests another processor core to stop when the system is shutting down by the first processor.

- **SI (Spurious Interrupt)**: Unwanted hardware interrupt e.g., when an interrupt that has been signaled to the processor and it is no longer required, or buggy hardware.

**Interrupt storm**: occurs whenever the operating system receives a large number of interrupts (from hardware) that consumes the majority of the processor’s time spurious signals, interrupt handler execution took too long, . . ., faulty drivers. The system becomes in *livelock*.
System Calls

Definition

**System Calls:** Constitutes an interface between the user (programs) and the services that are made available by an operating system.

- A systems call is a programmatic way in which a computer program requests a service from the operating system.
- Each operating system has its own system calls (e.g., Linux has 300, FreeBSD has 500, and Windows 7 has 700).
- System calls point to specific programs called routines that are written in C, C++, or assembly language.
- Operating systems provides an API (Application Programming Interface) as an interface between user programs and system calls.
System Calls

System calls can be grouped into six categories:

1. **Process control**: End, abort, load, execute, create process, terminate process, get/set process attributes, wait for a time, wait for an event, signal an event, allocate and free memory.

   *e.g.*, `exec()`, `fork()`, `wait()`, `exit()`, `getuid()`, `getgid()`, `getpid()`, `getppid()`, `signal()`, `kill()`, `acquire_lock()`, `release_lock()`.

2. **File management**: Create, delete, open, and close files, read, write on files, get/set files attributes.

   *e.g.*, `create()`, `open()`, `close()`, `read()`, `write()`, `lseek()`, `dup()`, `link()`, `unlink()`, `stat()`, `fstat()`, `access()`, `chmod()`, `chown`, `umask`, `ioctl()`.
# System Calls

1. **Device manipulation:** Request device, release device, read, write on device, get/set devices attributes.
   
e.g., `request()`, `release()`, `ioctl()`, `read()`, `write()`, `open()`, `close()`.

2. **Information maintenance:** get/set time or data, get/set system data, get/set resource attributes.
   
e.g., `time()`, `date()`, `alarm()`.

3. **Communication:** Create, delete a network connection, send and receive messages, transfer status information.
   
e.g., `pipe()`, `mmap()`, `shm_open()`, `gethostid()`.

4. **Protection:** Set ownership and access rights on a resource.
   
e.g., `chmod()`, `umask()`, `chown()`.
Hardware Interrupt

How Hardware events (keyboard presses, incoming network packets, mouse movement, ...) are escalated to running programs?

- Polling: When OS periodically queries each device to see if new information is available (simple but high latency and CPU cycle wasting).
- Interrupts: Device sends a signal to interrupt controller\(^1\) to request attention, CPU preempts running process to handle device request.

The period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt is called the interrupt latency.

After each (user mode) instruction is executed, the CPU checks whether the interrupt controller has any interrupt pending. If an interrupt is there, the current process is “suspended”, and the appropriate handler is executed.

\(^1\)PIC & APIC
Interrupts and Context Switching

Fetch-Execute Cycle

1. Fetch Instruction
2. Increment PC
3. Execute Instruction
4. Check if Interrupt Pending?
   - If No, go back to 1.
   - If Yes, go to next step.
5. Load PC from Fixed Memory Location (Interrupt Vector Table)
6. Push PC onto Stack
7. Switch to Kernel Mode
End