CPU Virtualization: The Process Abstraction

Prof. Patrick G. Bridges
How to provide the illusion of many CPUs?

- **CPU virtualizing**
  - The OS can promote the *illusion* that many virtual CPUs exist.
  - *Time sharing*: Running one process, then stopping it and running another
    - The potential cost is *performance*.
A Process

Comprising of a process:
- Memory (address space)
  - Instructions
  - Data section
- Registers
  - Program counter
  - Stack pointer

A process is a running program.
Process API

- These APIs are available on any modern OS.
  - Create
    - Create a new process to run a program
  - Destroy
    - Halt a runaway process
  - Wait
    - Wait for a process to stop running
  - Miscellaneous Control
    - Some kind of method to suspend a process and then resume it
  - Status
    - Get some status info about a process
Process Creation

1. **Load a program code into memory, into the address space of the process.**
   - Programs initially reside on disk in *executable format*.
   - OS perform the loading process *lazily*.
     - Loading pieces of code or data only as they are needed during program execution.

2. **The program’s run-time stack is allocated.**
   - Use the stack for *local variables, function parameters, and return address*.
   - Initialize the stack with arguments → argc and the argv array of main() function
Process Creation (Cont.)

3. The program’s heap is created.
   - Used for explicitly requested dynamically allocated data.
   - Program request such space by calling `malloc()` and free it by calling `free()`.

4. The OS do some other initialization tasks.
   - input/output (I/O) setup
     - Each process by default has three open file descriptors.
     - Standard input, output and error

5. Start the program running at the entry point, namely `main()`.
   - The OS transfers control of the CPU to the newly-created process.
Loading: From Program To Process

- **Disk**: Stores the program on disk.
- **Program**: Contains code, static data, and heap.
- **Memory**: Holds the process, including code, static data, and stack.
- **CPU**: Processed by the CPU.

**Loading**: Takes on-disk program and reads it into the address space of process.
Process States

- A process can be one of three states.
  - Running
    - A process is running on a processor.
  - Ready
    - A process is ready to run but for some reason the OS has chosen not to run it at this given moment.
  - Blocked
    - A process has performed some kind of operation.
    - When a process initiates an I/O request to a disk, it becomes blocked and thus some other process can use the processor.
Process State Transition

- **Running**
- **Ready**
- **Blocked**

States:
- **Descheduled**
- **Scheduled**

Transitions:
- I/O: initiate → Blocked
- I/O: done → Ready
- Scheduled → Running
- Descheduled → Ready
Data structures

- The OS has **some key data structures** that track various relevant pieces of information.
  - Process list
    - Ready processes
    - Blocked processes
    - Current running process
  - Register context

- **PCB (Process Control Block)**
  - A C-structure that contains information about each process.
Example) The xv6 kernel Proc Structure

```c
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip;    // Index pointer register
    int esp;    // Stack pointer register
    int ebx;   // Called the base register
    int ecx;   // Called the counter register
    int edx;   // Called the data register
    int esi;   // Source index register
    int edi;   // Destination index register
    int ebp;   // Stack base pointer register
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                 RUNNABLE, RUNNING, ZOMBIE };
Credits

- Disclaimer: This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University by Youjip Won. This lecture slide set is for the OSTEP book written by Remzi and Andrea at the University of Wisconsin.