The Abstraction: The Process

Sachidananda Urs

June 3, 2021

Abstract

We try to understand one of the basic absctractins, a process. Operating System provides the illusion of a nearly endless supply of said CPUs. We try to understand how the Operating System provides the illusion of many CPUs.

Introduction

Operating System provies an illusion of many CPUs by virtualizing the CPU. The concept of a process helps us understand how the Operating System virtualizes the CPU. CPU is shared between the processes by stopping a process, running another, so on...

But this is not free, but comes with a cost, “performance”. As the number of processes increase their running time increases as well, i.e each process runs slowly. This technique is called “time sharing of the CPU”.

We try to answer the following questions. *What constitutes a process and what is meant by its machine state?. Understand the the process API: create, destroy, wait, status, ... How is a process created? What are the process states? What are the data structures of a process?*

Getting back to the question *“How is CPU virtualized?”*

The Operating System uses low-level machinery and high-level intelligence to provide the CPU virtualization. i.e *Mechanisms* (low-level machinery) and *policies* (high-level intelligence).

1. **Mechanisms** include context-switching. We learn *what is a context switch?* and *How is it implemented?.* Context-switch gives the Operating System ability to stop running one program and start running another on a given CPU which is referred to as *time sharing.*

2. **Policies** are algorithms making some kind of decision within the Operating System. For example, if n programs are run simultaneously

which one is run first? second? so on ... The "Scheduling policy" in the Operating System makes this decision.

Linux supports the following scheduling policies: *FIFO*, *Round Robin*, *CFS - Completely Fair Scheduler*, *SCHED_DEADLINE*, *SCHED_OTHER* ... see sched(7) for more details.

Illumos supports *FSS - Fair Share Scheduler*, *Fixed Priority* ... (are there more?)

The Abstraction: The Process

Process is one of the most basic abstractions. We try answer the following questions:

1. Machine state of a process
   (a) What a process can read/update?
   (b) What parts of the machine are important for its execution?
      i. *Memory* that process can access (its address space)
      ii. *registers* (PC, SP, FP)
      iii. *Persistent storage devices* (List of files process has opened)

Process API

We try to understand the API provided by the Operating System to manage the processes, *create, destroy, wait, status* to name a few.

Process Creation: To create a process the OS must first find the program on disk and load it into memory (code and static data, this forms the address space of the process).

The loading of the program is done in two ways:

1. Eagerly – all program is loaded at once
2. Lazily – program is loaded as needed (see paging and swapping)

Then allocate memory for program's run-time stack and initialize the required variables.

Allocate memory for program's heap. Heap will be small at first and may grow to become very huge as the program runs.

Then the other initializations like setting up the input, output, error descriptors are done.

Finally the process is started, i.e at main. The OS tranfers the control of the CPU to newly created process, thus the program begins its execution.

Process States

In a very simplified view, a process can be in any of the following states

1. **Running** – Process is running on a processor executing instructions.
2. **Ready/Waiting** – At the moment process doesn't have a processor, OS has not given control of the processor.
3. **Blocked** Process is waiting for some other event. For example, process has initiated an I/O and waiting for it to finish, meanwhile processor is relinquished for another process.

Conclusion

In the subsequent notes we learn the process data structures and how they are used to manage the processes and how to manipulate them.