Operating Systems: Three Easy Pieces

§ 2. Introduction

KUMO B4 shuya

Running Program = Execute Instructions

Many millions of time every second the Processor …

- **fetches** an instruction from memory
- **decodes** it (figure out which instruction it is)
- **executes** it (it does the thing supposed to do)

A body of Software

- Above model is called **Von Neumann** model of computing
- A body of software is called the **operating system (OS)**, it is in charge of making sure system operates in an easy-to-use

Virtualization, Primary way the OS manage resources

- OS takes resources and transforms it into **virtual** form of itself.
- We call the technique **virtualization**.
- OS allows users to use virtual machine through **system calls**.
- Also saying that OS provides a standard library to applications.

OS is a resource manager

- Each of CPU, memory, disk is a **resource** of the system; it is thus operating system's role to manage those resources

2.1 Virtualizing the CPU

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main (int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];
    while (1) {
        Spin(1);
        printf("%s\n", str);
    }
    return 0;
}
```

Outputs

```
[~/dev/kumo/OSTEP/src/chapter2]
$ gcc -o cpu cpu.c -Wall
$ ./cpu "A"
A
A
A
^C
$ ./cpu A &; ./cpu B &; ./cpu C &; ./cpu D &
[1] 4285
[2] 4286
[3] 4287
[4] 4288
[~/dev/kumo/OSTEP/src/chapter2]
$ A
B
C
D
A
B
C
D
A
```

We have one processor but…

- It turns out that the system has large number virtual CPUs.
- When two programs want to run at a particular time, which run ?

-> It depends on a **policy** of the OS.

```
[~/dev/kumo/OSTEP/src/chapter2]
$ gcc -o cpu cpu.c -Wall
$ ./cpu "A"
A
A
A
^C
$ ./cpu A &; ./cpu B &; ./cpu C &; ./cpu D &
[1] 4285
[2] 4286
[3] 4287
[4] 4288
[~/dev/kumo/OSTEP/src/chapter2]
$ A
B
C
D
A
B
C
D
A
```

Physical memory is very simple

Memory is just an array of bytes

- To **read** memory specify an **address** to be able to access the data stored there.
- To **write** memory, also specify the data to be written to the given address.
- Memory accessed all the time when program is running.

2.1 Virtualizing the CPU

Code

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main (int argc, char *argv[])
{
    int *p = malloc(sizeof(int));
    assert(p != NULL);
    printf ("(%d) address pointed to by p: %p\n",
        getpid(), p);
    *p = 0;
    while(1) {
        Spin (1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);
    }
    return 0;
}
```

Outputs

```
$ ./mem
(5828) address pointed to by p: 0x7f9ba04002e0
(5828) p: 1
(5828) p: 2
(5828) p: 3
$ ./mem & ; ./mem & ;
[1] 5433
[2] 5434
(5433) address pointed to by p: 0x7f96944002a0
(5434) address pointed to by p: 0x7fb1ca4002e0
$ (5433) p: 1
(5434) p: 1
(5433) p: 2
(5434) p: 2
```

2.2 Virtualizing Memory

Result1 - single memory space

- It out out the address of memory and process identifier (PID).
- The newly allocated memory is at 0x7f9ba04002e0.

```
$ ./mem
(5828) address pointed to by p: 0x7f9ba04002e0
(5828) p: 1
(5828) p: 2
(5828) p: 3
```

Result2 - virtualizing memory

- Allocated same address
- Each program update the value, **virtual address space** make it possible.

```
$ ./mem & ; ./mem & ;
[1] 5433
[2] 5434
(5433) address pointed to by p: 0x7f96944002a0
(5434) address pointed to by p: 0x7fb1ca4002e0
$ (5433) p: 1
(5434) p: 1
(5433) p: 2
(5434) p: 2
```

2.2 Virtualizing Memory

We have one processor but...

- It turns out that the system has large number virtual CPUs.
- When two programs want to run at a particular time, which run ?

-> It depends on a **policy** of the OS.

```
[~/dev/kumo/OSTEP/src/chapter2]
$ gcc -o cpu cpu.c -Wall
$ ./cpu "A"
A
A
A
^C
$ ./cpu A &; ./cpu B &; ./cpu C &; ./cpu D &
[1] 4285
[2] 4286
[3] 4287
[4] 4288
[~/dev/kumo/OSTEP/src/chapter2]
$ A
B
C
D
A
B
C
D
A
```

A multi thread program

- Create two **threads** using *Pthread_create( )*
- Each thread starts running in a routine called *worker( )*
- The value of *loops* determines how many times each of the two workers will increments the shared counter in a loop.

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker (void * arg) {
    int i;
    for (i= 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int
main (int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: threads
<value>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);
    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);
    return 0;
}
```

Result1

- When the input value of *loops* is set to N, final output to be *2N*.

```
$ ./threads 1000
Initial value : 0
Final value : 2000
```

Result2 - higher values

- The program takes three instructions; load, increments, store.
- These instructions don't execute **atomically**, strange things can happen.
- It is this problem of **concurrency**.

```
$ ./threads 100000
Initial value : 0
Final value : 112807
$ ./threads 100000
Initial value : 0
Final value : 118738
```

2.4 Persistence

Needed to be able to store data persistency

- Such as DRAM store values in a **volatile** manner, when power goes away or the system crashes any data in memory is lost. So persistence is important.

- The hardware comes in the form of **input/output** or **I/O** devices, a **hard drive** is a repository for long lived information.
- **File system**; it is responsible for storing any files.
- **System calls** are routed to the part of the operating system called file system.

2.5 Design Goals

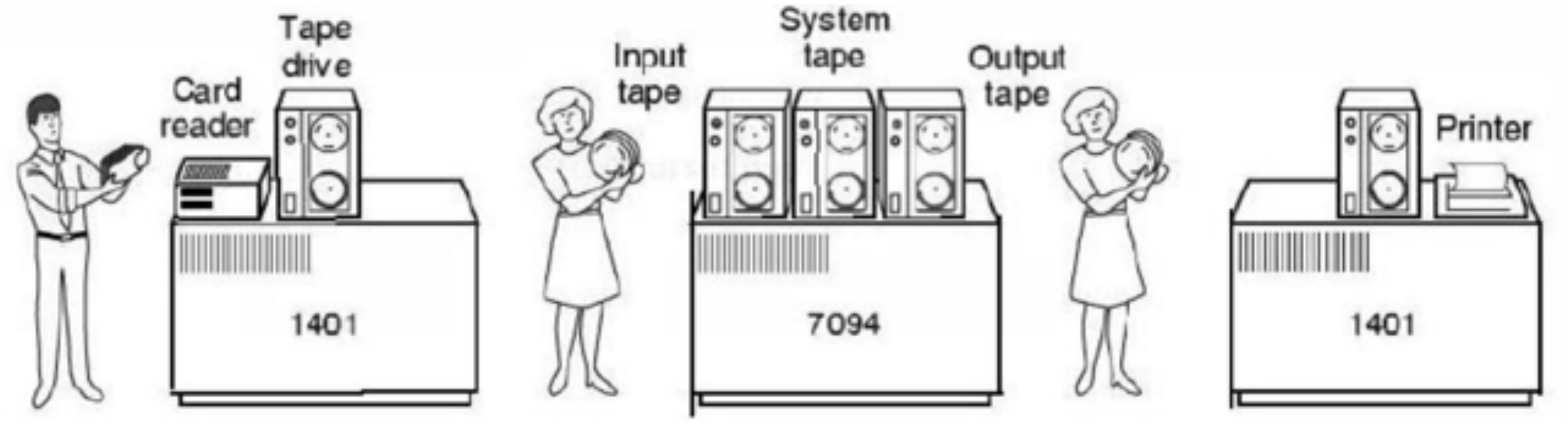
Finding the right set of trade-off is a key

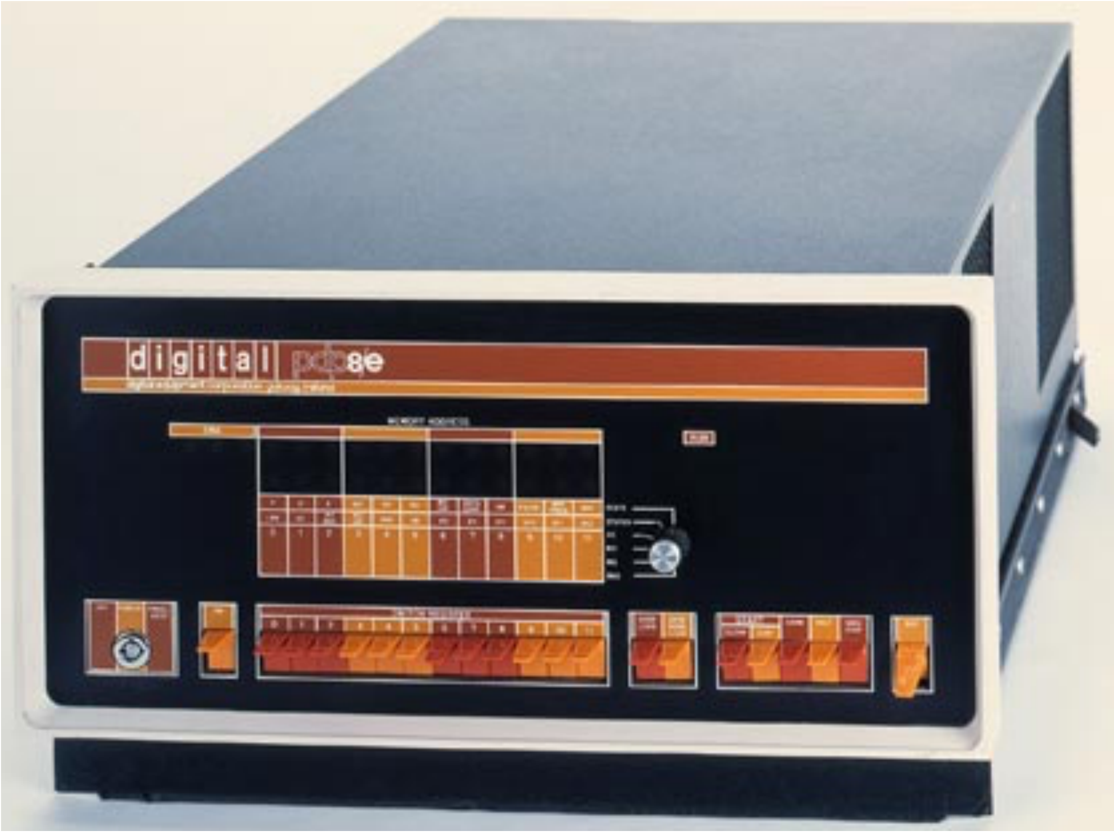
- **Abstractions**
  -> It is fundamental to everything we do in CS. Abstraction is a technique for arranging complexity of computer systems.
- **Performance**
  -> This can paraphrase to **minimize the overheads**. Virtualization make the system easy to use.
- **Protection**
  -> Isolating process from one another is key to protection.
- **Reliability**
  -> The OS must run non-stop.

2.5 Some History

昔の話はようわからん

- Early Operating System : Just Libraries
- Beyond Libraries : Protection
- The Era of Multiprogramming
- The Modern Era






パショコンの使いかちゃがむじゅかちい