Lecture 1: Introduction to Operating Systems

Operating Systems

Content taken from: https://www.cse.iitb.ac.in/~mythili/os/

Course Administration

- **Lectures:** Monday and Wednesday: 3 pm to 4:30 pm in C102 and C101 respectively.
- **Tutorials:** Wednesdays 8:30 am to 9:30 am in C201
- **Google Classroom:**

Google Classroom Code: **y6okogog**

  - All notices will be posted on google classroom only. Please ensure that you register on Google Classroom and are receiving all the notifications.
- **Office hours for instructor, TF and TAs:** TBD

Textbooks

- **Primary textbook:**
  - Operating Systems: Three Easy Pieces
  - https://pages.cs.wisc.edu/~remzi/OSTEP/
- **Other textbooks:**
  - Operating Systems Concepts
  - https://www.os-book.com/OS10/

Other details

- **Pre-requisites**
  - **Mandatory:** CSE102 Data Structures & Algorithms
  - **Desirable:** C Programming
- **Evaluation Components**
  - Quiz (4): 10%; best 3 out of 4
  - Assignments (4): 35%
  - Midsem: 20%
  - Endsem: 35%
- **Course pass criteria:** Get at least 30% in (midsem + endsem), i.e., at least 17 out of 55%.
- **Course Description:** Available on Techtree

Use of Generative AI (LLMs, ChatGPT, …)

- Use of Generative AI is strictly prohibited unless otherwise explicitly stated in the instructions of that particular assignment.
- Do not use them to generate code and submit it as your own code.

- You are free to use Generative AI constructively
  - For learning the material
  - Explanation of concepts
  - Get practice problems

Plagiarism policy

- Institute's plagiarism policy will be followed.
- It is available at https://www.iiitd.ac.in/academics/resources/academic-dishonesty.
- If you have any questions on the plag policy, please get back to me.

- Please do not indulge in it.
  - It is a waste of time for me.
  - It is a mental strain for you, along with the other multitude of things that come with a plag mark.

What is an operating system?

- Middleware between user programs and system hardware
- Manages hardware: CPU, main memory, IO devices (disk, network card, mouse, keyboard etc.)

What happens when you run a program? (1)

- A compiler translates high level programs into an executable (“.c” to “a.out”)
- The exe contains instructions that the CPU can understand, and data of the program (all numbered with addresses)
- Instructions run on CPU: hardware implements an instruction set architecture (ISA)
- CPU also consists of a few registers, e.g.,
  - Pointer to current instruction (program counter or PC)
  - Operands of instructions, memory addresses

What happens when you run a program? (2)

- To run an exe, CPU
  - fetches instruction pointed at by PC from memory
  - loads data required by the instructions into registers
  - decodes and executes the instruction
  - stores results to memory
- Most recently used instructions and data are in CPU caches for faster access

What does the OS do?

- OS manages program memory
  - Loads program executable (code, data) from disk to memory
- OS manages CPU
  - Initializes program counter (PC) and other registers to begin execution
- OS manages external devices
  - Read/write files from disk.
- OS supports concurrency
  - Provides support for concurrently executing threads within the same memory space and ensuring a correctly working program

OS manages CPU

- OS provides the process abstraction
  - Process: a running program
  - OS creates and manages processes
- Each process has the illusion of having the complete CPU, i.e., OS virtualizes CPU
- Timeshares CPU between processes
- Enables coordination between processes

Example 1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <assert.h>
5  #include "common.h"
6
7  int
8  main(int argc, char *argv[])
9  {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1);
17         printf("%s\n", str);
18     }
19     return 0;
20 }
```

Figure 2.1: **Simple Example: Code That Loops And Prints (cpu.c)**

Example 1

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
prompt>
```

Example 1

```
prompt> ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
...
```

Figure 2.2: **Running Many Programs At Once**

OS manages memory

- OS manages the memory of the process: code, data, stack, heap etc
- Each process thinks it has a dedicated memory space for itself, numbers code and data starting from 0 (virtual addresses)
- OS abstracts out the details of the actual placement in memory, translates from virtual addresses to actual physical addresses

Example 2

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "common.h"
5
6  int
7  main(int argc, char *argv[])
8  {
9      int *p = malloc(sizeof(int));                    // a1
10     assert(p != NULL);
11     printf("(%d) address pointed to by p: %p\n",
12            getpid(), p);                             // a2
13     *p = 0;                                          // a3
14     while (1) {
15         Spin(1);
16         *p = *p + 1;
17         printf("(%d) p: %d\n", getpid(), *p);        // a4
18     }
19     return 0;
20 }
```

Figure 2.3: **A Program That Accesses Memory (`mem.c`)**

Example 2

```
prompt> ./mem
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

Example 2

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
(24113) p: 4
(24114) p: 4
...
```

OS manages devices

- OS has code to manage disk, network card, and other external devices: device drivers
- Device driver talks the language of the hardware devices
  - Issues instructions to devices (fetch data from a file)
  - Responds to interrupt events from devices (user has pressed a key on keyboard)
- Persistent data organized as a filesystem on disk