CS 332/532 Systems Programming

Lecture 7

- Struct, OS-

Professor : Mahmut Unan – UAB CS

Agenda

- Structs
- Unions
- Operating Systems
- Unix Architecture

CS Networking Event

- 10/02/2024 Wednesday
- Prepare your resume

Structures & Unions

```
struct structure_tag {
 member_list;
} structure_variable_list;
```

A **struct** declaration defines a type. Although the structure_tag is optional, we prefer to name the structures we declare and use that name later to declare variables.

```c
struct company
{
    char name[50];
    int start_year;
    int field;
    int tax_num;

    int num_empl;
    char addr[50];
    float balance;
};
```

sizeof()

```c
#include <stdio.h>
struct date
{
        int day;
        int month;
        int year;
};
int main(void)
{
        struct date d;
        printf("%u\n", sizeof(d));
        return 0;
}
```

sizeof()

```
struct test1
{
        char c;
        double d;
        short s;
};
struct test2
{
        double d;
        short s;
        char c;
};
```

```c
#include <stdio.h>
struct student
{
    int code;
    float grd;
};
int main(void)
{
    struct student s1, s2;
    s1.code = 1234;
    s1.grd = 6.7;
    s2 = s1; /* Copy structure. */
    printf("C:%d G:%.2f\n", s2.code, s2.grd);
    return 0;
}
```

Dot notation vs Arrow notation

- In C, both dot (.) and arrow (->) operators are used to access members of a structure, but they are used in different situations.
- **Dot Notation (.):**This is used when you have a structure variable, not a pointer to a structure. It directly accesses members of the structure.
- **Arrow Notation (->):**This is used when you have a pointer to a structure. It dereferences the pointer and then accesses the member.

Dot notation

```c
#include <stdio.h>
#include <string.h>

struct Person {
    char name[50];
    int age;
};

int main() {
    struct Person person1;
    // Using dot notation to access members
    person1.age = 30;
    strcpy(person1.name, "Alice");

    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    return 0;
}
```

```
Name: Alice
Age: 30
```

Arrow operator

```c
#include <stdio.h>
#include <string.h>

struct Person {
    char name[50];
    int age;
};

int main() {
    struct Person person2;
    struct Person *ptr = &person2;  // Pointer to structure

    // Using arrow notation to access members
    ptr->age = 25;
    strcpy(ptr->name, "Bob");

    printf("Name: %s\n", ptr->name);
    printf("Age: %d\n", ptr->age);
    return 0;
}
```

```
Name: Bob
Age: 25
```

Unions

- Like a structure, a union contains one or more members, which may be of different types. The properties of unions are almost identical to the properties of structures; the same operations are allowed as on structures.
- Their difference is that the members of a structure are stored at *different* addresses, while the members of a union are stored at the *same* address.

```c

#include <stdio.h>

union Data {
    int i;          // 4 bytes
    float f;        // 4 bytes
    char str[20]; // 20 bytes
};

int main() {
    union Data data;  // Declare a union variable

    // Use sizeof to determine the size of the union
    printf("Size of union: %lu bytes\n", sizeof(data));
    return 0;
}

```

```
Size of union: 20 bytes
```

Sample Union usage

```c
#include <stdio.h>
#include <string.h>

union SensorData {
    int temperature;     // Temperature sensor in degrees Celsius
    float humidity;      // Humidity sensor in percentage
    char status[10];     // Status message like "OK", "ERROR"
};

int main() {
    union SensorData data;

    // Case 1: Using temperature sensor data
    data.temperature = 25;
    printf("Temperature: %d°C\n", data.temperature);

    // Case 2: Using humidity sensor data (overwrites temperature)
    data.humidity = 65.5;
    printf("Humidity: %.1f%%\n", data.humidity);

    // Case 3: Using status message (overwrites humidity)
    strcpy(data.status, "OK");
    printf("Status: %s\n", data.status);

    return 0;
}
```

```
Temperature: 25°C
Humidity: 65.5%
Status: OK
```

Operating Systems

- What is an operating system?
  - What stands between the user and the bare machine
  - The most basic and the important software to operate the computer
  - Similar role to that conductor of an orchestra
- It manages the computer's memory and processes, as well as all of its software and hardware.
- It also allows you to communicate with the computer without knowing how to speak the computer's language (hide the complexity from user)
- Without an operating system, a computer is useless.

https://edu.gcfglobal.org/en/computerbasics/understanding-operating-systems/1/

The Role of OS

- OS exploits the hardware resources of one or more processors to provide a set of services to system users
- OS manages secondary memory and I/O devices on behalf of its users
- In short,
  - OS manages the computer's resources, such as the central processing unit, memory, disk drives, and printers
  - establishes a user interface
  - executes and provides services for applications software.

OS

- A general –purpose, modern OS can exceed 50 million lines of code
- New OS are being written all the time
  - E-book reader
  - Tablet
  - Smartphone
  - Mainframe
  - Server
  - PC
  - …..

Why to learn OS?

- To be able to write concurrent code
- Resource management
- Analyze the performance
- To fully understand how your code works
- ….
- In short,
  - this class isn't to teach you how to CREATE an OS from scratch, but to teach you how an OS works

Unsolved problem

Operating systems are an unsolved problem in computer science. Because;

- *Most of them do not work well.*
  - *Crashes, not fast enough, not easy to use, etc.*
- *Usually they do not do everything they were designed to do.*
  - *Needs are increasing every day*
- *They do not adapt to changes so easily.*
  - *New devices, processors, applications.*
- ……

Operating System Services

- execute a new program
- open a file
- read a file
- allocate a region of memory
- get the current time of day
- so on

UNIX Architecture

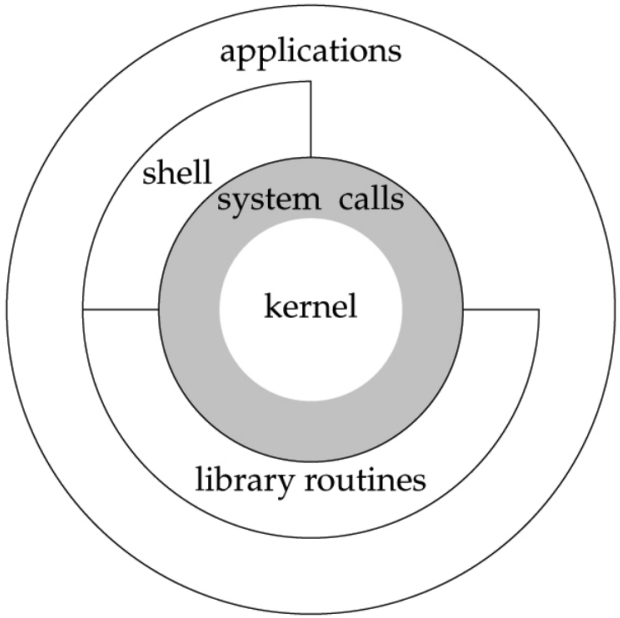

Figure 1.1 Architecture of the UNIX operating system

References

- C From Theory to Practice - 2nd edition, Nikolaos D. Tselikas and George S. Tselikis
- https://www.guru99.com/c-dynamic-memory-allocation.html
- https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/
- https://www.elprocus.com/unix-architecture-and-properties/