CS 332/532 Systems Programming

Lecture 7

- Dynamic Memory Allocation, OS-

Professor : Mahmut Unan – UAB CS

Agenda

- Memory operations
- Structs
- Unions
- Operating Systems
- Unix Architecture

Static Memory Allocation

- In static allocation, the memory is allocated from the stack.
- The size of the allocated memory is fixed; we must specify its size when writing the program and it cannot change during program execution.
- For example, with the statement:

```
float grades[1000];
```

Dynamic Memory Allocation

- In dynamic allocation, the memory is allocated from the heap during program execution. Unlike static allocation, its size can be dynamically specified.
- Furthermore, this size may dynamically shrink or grow according to the program's needs.
- Typically, the default stack size is not very large, the size of the heap is usually much larger than the stack size.

malloc()

```
void *malloc(size_t size);
```

The `size_t` type is usually a synonym of the **unsigned int** type.

The size parameter declares the number of bytes to be allocated.

If the memory is allocated successfully;

`malloc()` returns a pointer to that memory, NULL otherwise.

Check the following functions

`realloc()`

`calloc()`

`free()`

`memcpy()`

`memmove()`

`memcmp()`

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

int main()
{   int *ptr,n,i;
    /* the number of array elements */
    printf("How many elements?:\n");
    scanf("%d",&n);

    ptr = (int*)malloc(n * sizeof(int));

    if (ptr == NULL) {
        printf("Memory allocation was NOT successful.\n");
        exit(0);
    }
    else {
        printf("Memory  allocation was successful.\n");
        for (i = 0; i < n; i++)
            ptr[i] = (i+1) * 10;

        for (i = 0; i < n; ++i)
            printf("%d, ", ptr[i]);
    }
    free(ptr);
    printf("\nMemory deallocation was successful.\n");
    return 0;
}
```

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

int main()
{   int *ptr,n,i;
    /* the number of array elements */
    printf("How many elements?:\n");
    scanf("%d",&n);

    ptr = (int*)malloc(n * sizeof(int));

    if (ptr == NULL) {
        printf("Memory allocation was NOT successful.\n");
    return 0;
}
```

```
How many elements?:
8
Memory  allocation was successful.
10, 20, 30, 40, 50, 60, 70, 80,
Memory deallocation was successful.
```

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.

```
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;
}
```

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 sample
{
    char ch;
    int i;
    double d;
};
int main(void)
{
    union sample s;
    printf("Size: %u\n", sizeof(s));
    return 0;
}
```

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

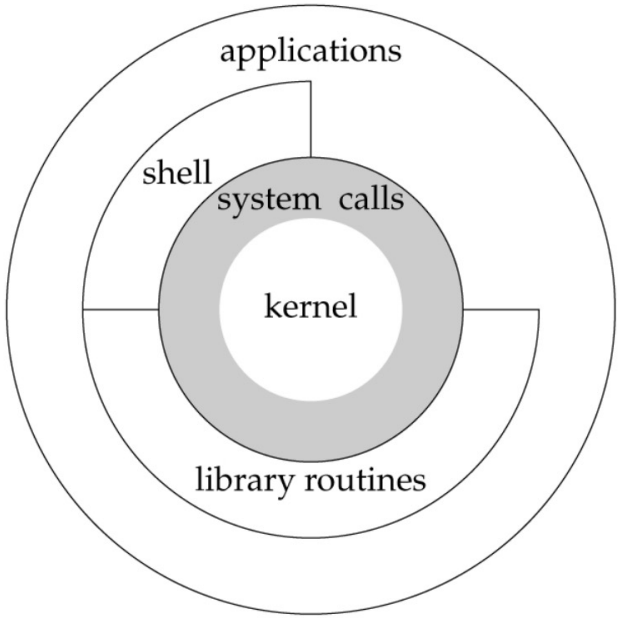

Figure 1.1 Architecture of the UNIX operating system

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/