Proposal - Group 37

Introduction

1. We go for topic 1 - A Simulated Operating System.
2. The components we are going to implement are

- Memory Management (Random Access Memory/RAM, Read Only Memory/ROM, Memory Allocation, Memory Release)
- Task Scheduling & Processing (Processes and Threads, Scheduling Algorithms, Scheduling simulation/SS)
- File System
- Exception and Security
- User Interface (Qt based GUI)

Related Work

1. We all vote for operating system because we use Windows and MacOS operating systems but never have thought about how to construct one, thus it attracts us very much.
2. Google and the resources that come with Qt Creator.

Our Work

I. The Work Principle

<u>Differences between the basic system and the real OS</u>

1. Stored in external environment

Our system looks like a virtual machine and runs in an external environment. In other words, all our parts are stored in the real memory. The biggest difference between our system and the real system is that most of the instructions in our system need to be operated by external systems. The basic system we formed does not have permission to directly drive the CPU to do the calculation, so we can only take the use of the external system to drive CPU to process related instructions.

For example, to modify the content of a specific byte in analog RAM, the function of our modified instruction has undergone the progress as

store in analog ROM >> read to analog RAM >> transfer to external real memory >> become an executable C ++ instruction >> pass to the CPU for operation and execution >> CPU then modifies the specific byte

2. Using analog hardware (RAM, ROM)

However, because our memory is just a simulated 2D array, it can only perform relatively simple operations, and can not perform too complicated operations. Because there is no underlying assembly language to complete the most basic computer instructions, we cannot load the relevant hardware drivers. In addition, because our memory and disk is a completely virtual 2D array, the instructions loaded in analog RAM cannot access the hardware drivers of external systems.

In order to simulate the process of real system operation, we thought of the following methods to mimic the operation management process of the operating system reading and writing virtual memory.

Methods of the simulation

1. Using ASCII encoding

In order to find a one-to-one correspondence between the binary 0 and 1 storage format and natural language, we have adopted the internationally accepted ASCII encoding standard. For example, the encoding of the character 'a' is 97, then converted into a binary array is "01100001". We will use this 8-bit binary format array as one byte. Because ASCII has a total of 128 = 2 ^ 7 encodings, then a 7-bit encoding format is sufficient. But in order to follow the operating habits of computer systems, we use an 8-bit storage format. That is, each byte can store a separate character.

Because our operating system is not very complicated, we did not consider reading and writing Chinese characters in the more complicated Unicode encoding. If you use the Unicode-8 encoding format, characters in different formats need 1-4 bytes to complete the one-to-one correspondence, which causes considerable trouble for our RAM index.

2. The storage location of the system is analog ROM

Because the code of this project is also completely composed of standard characters in the ASCII range, we can store the code except the initialization part in the analog ROM. Every time the system is turned on, all instructions in the operating system are stored in the analog ROM in the form of an 8-bit binary array Then instructions are read into the analog RAM, been

translated into executable C++ according to the ASCII encoding standard. The C++ code is passed to real memory and then passed to the CPU for execution.

3. Analog BIOS

We can imitate the BIOS part of the real operating system and define the system initialization part as the BIOS of our system. When we simulate the boot process, that is, the process of starting to run our system in the external environment, only a small amount of codes of the basic system will be initialized in the external system. We call this part of the code an analog BIOS. The analog BIOS code is very intuitive and is displayed directly in the external environment.

After the CPU in the external environment reads the analog BIOS, the analog BIOS initializes the analog RAM and the analog ROM, that is, our simulated hardware. This process simulates the process of loading the hardware driver by the BIOS in a real system.

That is, all the code parts except for the analog BIOS are packaged in analog ROM, so that the real system's storage structure can be simulated: stored in the hard disk and loaded in memory. Therefore, our analog ROM will leave a sector to store our operating system.

4. Establish a valid path in analog ROM

Each instruction will eventually become, to modify the Byte at a specific location in the virtual RAM.

The difference between our analog RAM and analog ROM is that analog RAM will be restarted every time the program runs, restoring the state where all bits are 0; but the contents of the analog ROM will be saved.

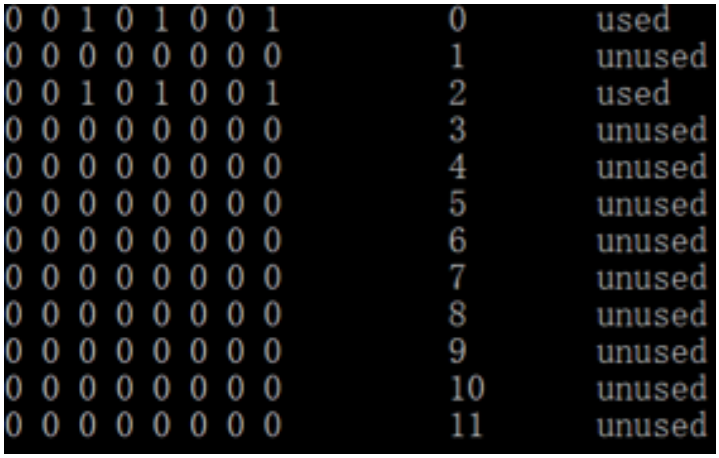
In order to read the contents stored in ROM, we need to establish a valid storage path in analog ROM.

II. Memory Management

Basic Idea
The simulation process of the memory management in our Operating System can be divided into the following four parts:

1.Memory Establishment

Define an m*n*8 3D-array memory to be our simulating “memory”. In this case, each element in the 3D-array can represent a “bit”, then each row of the array (8 bits) represents 2 “bytes” (which can also fit in the 8-number binary decoding rule of ASCII standard). For instance, when m = 1 & n = 512, the 3D-array itself can be a simulation of 1KB internal memory in the normal Operating System. This 1KB memory can also be call a “page” for future convenient. Combining 1024 these kind of “page”, we eventually get 1MB space.



1.Information Collection

Giving an index to each of the row (byte), and define an n-element array usage which contains the usage information of each byte (1 for used, and 0 for empty). Define an n-element array address which used for future symbols assigning. Then we are able to define an infomation-bag for each byte which contains roughly three parts of information: {content, index, usage}. The “content” represents the actual content stored in this byte memory; “index” represents the address symbol; “usage” indicates whether this byte was used or not.

2. Memory Allocation

Define the dynamic memory allocating function allocation(), in each time of allocation, searching for a continuous unused sequence of memory according to the size required. Then turn the corresponding elements in the usage array from 0 to 1, marking these bytes with the same address symbol. After that, return the address of the first byte.

3. Memory Release

When the content of any objects need to be deleted, using the specific address symbol in address defined before to find the target bytes. Remove these items in memory, and change the corresponding position in usage from 1 to 0.

Functional Partitioning

Usually, a complete storage module will contain mainly two parts: RAM (Random Access Memory) and ROM (Read Only Memory). In our own Operating System, we are going to simulate the memory management procedure from these two parts.

1.RAM [20% space of memory]

When the Operating System starts running, initialize the space of RAM to empty. Loading basic parameters and file data which have already stored in ROM. Once the user has started a process, assigning a specific region of the RAM by the estimated scale of the process, then doing the information read-in and read-out according to user's instructions. At the same time, keep interacting with CPU for data analysis and data processing. Finally, when the process is ended, clear all the data belongs to this specific process.

1.ROM [80% space of memory]

ROM is basically used for storage of relatively permanent information and file data. We are going to construct our ROM with exactly the same structure of RAM but totally different accessibility. The read-in and read-out of ROM will undergoes a strict method. For instance, when the user wants to save the file, the information will be stored in ROM. Laterly, he can use the "delete" method to clear the data in ROM.

III. Process and Scheduling

Process

1.Process memory

The process memory is divided into four part:

a.The TEXT Section is made up of the complied program code, read in from the non-volatile storage

b.The Data section is made up the global variables and static variables, allocated prior to execution the main.

c.The HEAP is used for the dynamic allocation.

d.The STACK is used for local variables.

e.Process states:

f.NEW: create a process.

g.READY: process is waiting to be executed.

h.RUNNING: instructions are being executed.

i.WAITING: process is waiting for something occur (e.g. an I/O completion or reception of a signal).

j.TERMINATED: process has finished execution.

k.Process control block, enclosing all the information about a process, which contains the following:

l.Process states

m.Process ID and its parent process ID.

n.Scheduling information: such as the pointers to the scheduling queues.

o.Memory Management information.

p.Accounting information.

q.I/O status information.

Process Creation

In Linux, for example, all processes have a parent process except the initial process. All processes are copied or cloned from its parent process. A process can create its child process through an appropriate system call. Every process has a process identifier called PID and its parent process identifier called PPID is also stored for each process.

Process termination

A process can making a specific system call to request its own termination and return an integer. The return int is typically 0 if the process is successfully completed and return int if there are any problems.

A process may also be terminated in the following situation:

a.The system cannot give this process necessary resources.

b.In response to a command to kill this process.

c.Its parent process can kill this process if it is no longer needed.

d.If its parent process terminated, this process may be terminated.

Process scheduling

For a real operating system, it is important to schedule the processes. Because we want to keep the CPU busy all the time, and to minimize the response time of every process.

The act of determining which process is in the ready state should be moved to the running state is process scheduling.

There are two major categories of scheduling, which are Non Pre-emptive Scheduling: when the current process gives up the CPU voluntarily and Pre-emptive Scheduling: when the operating system decided to execute another process, pre-empty the current executing process.

In order to scheduling processes, we need queues. All processes, after entering into the process are stored into the Job queue. Processes in the ready state is

stored into the Ready queue. If a process needs devices signals, the process will be placed in the Device queue.

A new process is put in the Ready queue and waits to be selected for execution.

If the process is selected for execution, one of the following situations will happen:

The process has an I/O request, then placed in I/O queue. After getting the I/O signals then put back in ready queue.

The process could create a child process and waits for child process's termination, and then put back in ready queue.

The process could be removed for running state as a result of an interrupt, then put back into ready queue.

All processes are in this cycle, until they are terminated.

Since we do not control physical CPU directly, we will store the code of programs into our rom and let the outside system to read and execute our application programs.

For process, the problem we are going to solve is how to manage the memory for different processes to store data and status.

In order to show the details about processes scheduling. We can write a function to show the statistic results (running time, waiting time, etc.).

IV. Task models / Multitasking

Scheduling algorithm:

Considering the processes in our OS are not complicated, to achieve the goal of "Multitasking", we decide to use the Round Robin Scheduling algorithm.

Principle: Given a time slice t=10ms, the first task A in the waiting queue will be processed by the CPU for t (10ms). If task A is finished within 10ms, then the next task in the head of the waiting queue will be processed. If 10ms is not enough to finish task A, the "program counter" will record which line we have already done. Then Task A will be added to the end of the waiting queue. When processing task A next time, the CPU will go on from the number "program counter" recorded. By RR algorithm, with the small t, we can create a false appearance that we are processing two programs at the same time.

Method: We can use sleep () function to forgive the demand of one Task for the time slice. If we only have two tasks to achieve "multitasking", I think using sleep () function will be easy.

V. Graphical User Interface

Solution1 - Text-based User Interface
How do we do it?
Step 1: Type in commands on a command line using a keyboard…

What does the result look like?

Solution 2 - QML / Other Interface markup languages
How do we do it?
Step 1: Making use of existed libraries and packages to design an OS-like window which is only used for display. (Language: QML)
Step 2: Linked other components (applications, file system, etc.) with specific buttons on the pre-designed OS-like window.
Step 3: Test and debug, see if the integration is working, efficient, and robust.

What does the result look like?

What does the code look like?

Solution 3 - Web-based GUI

- C++ Web Development Framework; embeddable HTML / CSS engine with C / C++ API
- CppCMS
  * It is designed and tuned to handle extremely high loads.
  * It uses modern C++ as the primary development language in order to achieve the first goal.
  * It is designed for developing both Web Sites and Web Services.
- Awesomium
- librocket

Solution 4 - Qt GUI widgets

Strategy & Implementation

- Coding the user interface of simulated operating system using pure C++ code
- Making use of Qt GUI libraries and widgets
- Simplicity > Functionality > Appearance at the first stage

Sample result

VI. Exception

Exception System: Stop the execution when errors occur. Try to fix the problems and possibly reset the whole system if a fatal error (i.e. a triple fault) happens.

==> expectation:
Controllable error: stop the execution and fix the problem
Fatal error: reset the whole system

In general, an exception breaks the normal flow of execution and executes a preregistered exception handler.

Exception and interrupt handling

Overview:

When exception or interrupt occurs, execution transition from user mode to kernel mode where the exception or interrupt is handled.
Then when the exception or interrupt has been handled execution resumes in user space.

Details:

1.While entering the kernel, the context (values of all CPU registers) of the currently executing process must first be saved to memory.

1.The kernel is now ready to handle the exception or interrupt:
    1.determine the cause of the exception or interrupt
        (The information of cause can be stored in registers or at predefined addresses in memory)
    1. the exception or interrupt
        For instance,
        If it was a keyboard interrupt, then the key code of the keypress is obtained and stored some where or some other appropriate action is taken.
        If it was an arithmetic overflow exception, an error message may be printed or the program may be terminated.

1.When the exception or interrupt have been handled, the kernel performs the following steps:
    3.1 select a process to restore and resume

The kernel may choose to resume the same process that was executing prior to handling the exception, or resume execution of any other process currently in memory.

3.2 Restore the context of the selected process

The context of the CPU can be restored by reading and restoring all register values from memory.

3.3 Resume execution of the selected process

The address was saved by the machine when the interrupt occurred, so it is simply getting this address and make the CPU continue to execute at this address.

(Note: CPU context = CPU state = values of all the register)

Security

Authentication

It refers to identifying each user of the system.

Username/ password: user need to enter a registered username and password with OS to login into the system

VII. Application (Examples)

- Clock:

To fully use our 2D-RAM, I think we can get the local time in BIOS step. It means that in BIOS, we initialize the RAM to zeros and then given an array like this:

2020(year)		2(month)		28(day)		16(hour)		37(min)
Size:10-bit		4-bit		5-bit		5-bit		6-bit

Every second, it will add 1 and automatically calculate the carry-bit.

By using sleep (1000) function, we can get the exact time of how long the OS is be on.

The clock will “cout” the current time every 1000ms, and in the 1000ms when clock is sleeping, we can do other processes. This is the easiest way to achieve multitasking in our OS.

- Mine clearance

Given a 8*8 array based on our RAM, randomly change 8 unit to be 9 (9 represent Mine), others to be 1 (1 represent safe area).

Easy edition: Without flag, choosing by keyboard input.

Advanced edition: With flag, choose area by mouse click, left button and right button represent choose and flag.

Schedule

Milestones:

- First draft of proposal - Feb 24th 2020
- Second draft of proposal - Feb 29th 2020
- Final draft of proposal - Mar 3rd 2020
- First draft of code package - Apr 15th 2020
- First draft of report - Apr 17th 2020
- Second draft of code package - Apr 29th 2020
- Second draft of report - May 1st 2020
- Final code package - May 6th 2020
- Final report - May 8th 2020
- Code package & report Submission - May 12th 2020

Team

Team members

王天石 117010265
林仲豪 117010161
周亚楠 115010302
范智渊 117010059
王泽伟 117010274
王峻强 117010258 (Representative)

Work distribution

王天石 - Memory allocation
林仲豪 - Task models
周亚楠 - Main function
范智渊 - scheduling
王泽伟 - exceptions and security
王峻强 - User Interface, applications

*Note: The following work distribution is just for reference. Our team members have excellent team working sense and we are going to tackle all the potential problems together as a whole.