Butcher Operating System Technical Report

John Chu

Introduction/Overview

The Butcher Operating System is a 16-bit operating system created for use in Operating System course kernel project. This project simulates a basic operating system and is capable of executing the following implementations:

- Bootloader program launched on the floppya.img disk image to initiate Butcher Operating System kernel.
- Basic printing functions to print out "Hello World" using putChar function and putInMemory function.
- Interrupts generated by software to invoke the BIOS routine which reads the user-typed characters and sectors from the disk and displays the results via the screen.
- System Calls specifically oriented to user-defined Interrupt Service Routine to provide OS services.
- A file system to perform the basic I/O operations such as reading, writing, printing, and deleting the file on the disk.
- execution and termination of user programs.
- A command line shell with the commands (*dir, type, copy, delete, execute, ps, and kill*) that provides Graphical User Interface to perform administrative functions.
- Multiprogramming based on time sharing property and round-robin scheduling algorithm and basic memory management.

All functionalities in Butcher operating system are designed to operate as closely as an open source operating system such as Linux, Mac OS, and Window OS.

Setup Process

In order to set up and use Butcher operating system, following steps are required.

Step I. Download virtual machine with Mac High Sierra 10.13.vmdk version. Also, download Mac High Sierra 10.13.iso file and launch them on the virtual machine.

Step II. Install XQuartz software application to run the bochs program that enables to display the result of the operation of Butcher operating system.

Step III: Execute the XQuartz application and go to the directory project 5 using the command "cd /Desktop/Comp354Projects/project5"

Step IV: Run the compileOS.bash using *"./compileOS.bash"* on XQuartz terminal which contains a collection of command lines capable of launching the bootloader and user programs, compiling the programs and files, and linking the object files and placing the resulting executable files on the disk.

Implementation

As briefly mentioned in the introduction part, Butcher Operating System contains diverse rudimentary utilities to behave as an operating system throughout five projects conducted so far. For each project, following function implementations are used to construct the Butcher Operating System kernel:

- Project 1: The goal of project 1 is to acquaint myself with the boot process while also providing software tools and a simulator for future projects. I wanted to create a shell script that would run the OS kernel and display out "Hello World" on the visual memory screen. The compileOS.bash shell script had command lines to assemble and load the bootloader, create the disk image, compile and assemble the kernel, copy the kernel into the disk, load the executable kernel file by linking kernel files, and run the bochs program under the Butcher operating system. The putInMemory and putChar methods are then used by this operating system to print "Hello World" and display it on the video memory. At first, the putInMemory method simply inserts a single character into the memory address determined by the operating segment and address parameters. In addition, the putChar method extends the capability of putInMemory. The putChar function uses two lines of putInMemory routines to store a character at (row, col) in the video memory, with colors for the background and font. The putStr function is built using those two functions, and it greatly improves the functionality of the putChar function by printing each colored character and backdrop using an iteration method.
- Project2: This project aims to be able to use the system call interface, which enables user programs to request services from the operating system. Kernel.asm also has interrupt functionality to help with the previously mentioned capabilities. The interrupt function can successfully request a system call from the operating system in two ways: by using the BIOS Routine or by developing a system call interface. In order to print a character, read a user-input character, or read a disk sector, BIOS routine interrupts required a certain entry number. In project 2, the functions are implemented utilizing interrupts 0x10, 0x13, and 0x16, and printString, readChar, readString, and readSector functions are defined using those BIOS Routines. A user-defined system call interface is also being developed. The Interrupt Service Routine is responsible for providing OS services. Project 2 does this specifically with the kernel.asm file's makeInterrupt21 and handleInterrupt21 functions. When the handleInterrupt21function is called, the makeInterrupt21function sets up the interrupt 0x21 vector so that interrupt21ServiceRoutine can run. The interrupt 0x21 Interrupt Service Routine then delivers OS services ranging from BIOS Routines to yet-to-be-defined future functionality. ax, bx, cx, and dx are the four arguments of handleInterrupt21. The value of ax will determine which OS service the users want in this situation.
- Project 3: The third project focuses on writing routines to read files into memory and run applications. This will greatly aid in the creation of the file system required to keep track of the files on the disk. To facilitate the building of the Butcher operating system, project 3 creates a rudimentary shell program that executes other programs and prints text files, as well as a user library file that offers wrappers around the system call interface built in project 2. Project 3 begins by adding a few command lines to *compileOS.bash*, which adds file system and a test file to the disk image. Understanding what Disk Map and Disk Directory do and where they are situated is critical in this project. Disk Map keeps track of which sectors are vacant and which sectors are assigned to files, whereas Disk Directory provides the name of each file as well as the sectors assigned to it. Each sector 1,2 has a disk map and a disk directory. Following that, a file reading function is created using the

newly constructed file system. Similarly, loading and executing user programs generated by users using a few command lines and a file loading program is also possible. Furthermore, terminating a user-executed program allows the user program to relinquish control to the operating system. All of the project 3 features are defined in handleInterrupt21 and can be invoked using interrupt21ServiceRoutine. A command line shell application is written to read and recognize keyboard input from users and make relevant system calls in order to accomplish the functions stated above. Users can, for example, insert type <file> to visually inspect the contents of a file. Users can also write execute <file> to run the file loaded on the disk image. By calling interrupt 0x21, a user library is built to make it easier to employ functions that serve as wrappers for system calls.

- Project 4: Project 4 aims to add various additional commands to the shell software, as well as implement features for removing and writing files. Project 4 begins with the addition of a new BIOS procedure, the writeSector function, which supports the writing of a file. Following that, functions to delete and write files are defined in order for the operating system to completely act as a single-process operating system. The handleInterrupt21 function defines the writing and deleting file functionalities. This is because the shell application requires handleInterrupt21 from the user library to easily perform commands like delete <file>, copy <src> <dest>, and dir. In addition, a text editor application was developed that successfully performs the functionalities listed above, similar to the nano program in the Linux operating system.
- Project 5: Project 5 allows a single-processing operating system to be extended into an OS that can execute multiple processes simultaneously. Memory management and time-sharing properties are used to implement this crucial feature. Data structures such as Process Control Block and Ready Queue are provided to realize those two features. The status of the process, segment, stack pointer, name, and pointers to previous and next Process Control Blocks are all contained in a Process Control Block. Ready queue is also a data structure for storing a list of Process Control Blocks that are ready to be scheduled. Butcher operating system improves to multi-process operating system by conducting process management using these data structures. This is linked to starting new processes, dealing with timer interrupts, and terminating processes. The memory management system is used to locate a free memory segment in which to load the program and place the Process Control Block into the ready queue. When a timer interrupt occurs, the context of the currently running process is saved, and a new process from the ready queue is selected to execute. When a previously running process in the ready queue is rescheduled, the context of the currently running process is restored. Then, memory segment and Process Control Block release when the process terminates. Concurrent execution of multiple processes can be tested by running the user program 3 that the Butcher operating system created specifically for project 5 on a shell program and typing the "ps" command while both user program 3 and shell program are running. All the detailed implementations and explanations for all five projects are provided https://github.com/JuheonChu/OS-kernel.

User commands supported

Butcher operating system contains 7 user commands defined in shell script: type <file>, execute <file>, delete <file>, copy <src> <dest>, dir, ps, and kill . Details of each command are as follows:

- type <file>: type <file> command prints the contents of an input file on the screen. The Butcher operating system prints the message "File not Found!" if a file is not found.

- execute <file>: execute <file> command loads the input file into the memory and executes it. If a program executable file does not exist, then this will print out “Program does not exist.”
- delete <file>: delete <file> command deletes the input file that was loaded in the memory. If the user input file does not exist, then delete <file> will not do anything.
- copy <src> <dest>: copy <src> <dest> command copies the source file and write it to a new destination file by reading the source file and writing a destination file. If a destination file already exists, the destination file is overwritten. If there is no free directory or sectors to write the destination file, the Butcher operating system will print "No empty location available for the file!" or "Not enough space in the directory!".
- dir: The dir command lists all of the files in the disk directory. This is accomplished by successfully reading sector 2, which contains the disk directory.
- ps: ps command displays a list of the programs that are currently running. This command is useful to verify if the multiprogramming is successfully performing.
- kill : kill command kills a process whose segment equals the input segment index.

Bugs and Limitations

While constructing Butcher operating system, there are three difficulties to face.

1. writeFile function: writeFile function for Butche operating system can only successfully write a file named with five characters. Furthermore, overwriting the contents of a file whose name already exists in the directory fails.

2. printInt function: For integers that have 6 or more digits are not printed out correctly. Also, some hexadecimal numbers are failed to be printed.

Conclusion

In a nutshell, the Butcher operating system provides OS services. Directly designing the features of real-world operating systems like Linux and Mac OS at first place in person is imperative. From basic BIOS operations to a user-oriented shell script program, we can now see how the operating system plays an important role in interacting with every component of the computer. Although some constraints must be addressed, they will be addressed by the end of the semester. Constructing an entire operating system strengthened my muscle memory for interacting with theoretical concepts acquired in class and improved my C language skill in general. Personally, improving my coding skills through this project has been one of my most rewarding experiences.

Reference

(2022). S22-project1-description. In A. Instructor (Ed. Professor Siddiqui, Farhan), *Comp 354: Operating System*. Dickinson College

(2022). S22-project2-description. In A. Instructor (Ed. Professor Siddiqui, Farhan), *Comp 354: Operating System*. Dickinson College

(2022). S22-project3-description. In A. Instructor (Ed. Professor Siddiqui, Farhan), *Comp 354: Operating System*. Dickinson College

(2022). S22-project4-description. In A. Instructor (Ed. Professor Siddiqui, Farhan), *Comp 354: Operating System*. Dickinson College

(2022). S22-project5-description. In A. Instructor (Ed. Professor Siddiqui, Farhan), *Comp 354: Operating System.* Dickinson College