NachOS
Project: M1-MoSIG

Armando Ochoa, Jing Han, Luan Pham, Mina Cesar, Shubham Khandelwal

January 15, 2015

Contents

1 Introduction

Nachos(Not Another Completely Heuristic Operating System) is a teaching operating system where we get to implement missing functions that allows us to understand how an operating system works. The duration for this project is one month.

The remaining report is organized as follows. Section 2 describes about the Nachos Kernel Features, Section 3 includes the User-Documentation, section 4 have details of internel design of NachOS. Section 5 and 6 contains testing description and the issues related to implementation. Our work organization is written in section 7, Section 8 have the conclusion of the work.

2 Our Kernel Features

Here are some of the Kernel Features which NachOS kernel have:

- Console Input - Output
- Multi-threading

2.1 Interesting Features

3 User Interface

3.1 I/O System Call Description

Exit-

- Convention : void Exit(int **status**)
- Description : Exit is a function which can be used to quit the process.

Halt-

- Convention : void Halt()
- Description : Halt system call can be used to shut down/power off the system.

PutChar-

- Convention : void PutChar(char **c**)
- Description : PurtChar is a function used to write a character **c** to the simulated standard output stream.

GetChar-

- Convention : char GetChar()
- Description : GetChar is a function used to read the next character from simulated standard input stream and returns a char or EOF on the end of the input or nothing.

PutString-

- Convention : void PutString(char*__c__)
- Description : PutString is a function used to write a string **c** to the simulated standard output stream.

GetString-

- Convention : void GetString(char*__buffer__, int **n**)
- Description :GetString is a function used to read a string with size **n** from the simulated standard input stream into a buffer pointed by **buffer**.

PutInt-

- Convention : void PutInt(int **num**)
- Description :PutInt is a function used to write an integar **num** to the simulated standard output stream.

GetInt-

- Convention : void GetInt(int ***num**)
- Description : GetInt is a function used to read the next integar from the simulated standard input stream into a variable pointed by **num**.

UserThreadCreate-

- Convention : int UserThreadCreate(void f(void *arg), void *arg)
- Description : UserThreadCreate will create a new thread in the calling process, the new thread starts execution by invoking a routine named **threadfunction**, **arg** is passed as the sole argument of routine **threadfunction**.After creation of a new thread, it will return 0 to represent success or -1 to failure.

UserThreadExit-

- Convention : void UserThreadExit()
- Description : UserThreadExit will terminate the calling thread explicitly, which will make called thread wait until the calling thread terminate.

4 Internal Design

4.1 Thread Process Model Description

/*insert here about threadcreate/threadexit and halt!!!*/

5 Test Description

All our test for system call are based on functionality verification.

5.1 I/O System Call Test

1 - PutChar Test Script:
void print (char c, int n) int i; for (i = 0; i ¡ n; i++) PutChar(c+i);
Input: Characters to be displayed (abcd) Output: abcd
2 - char GetChar(): Test Script :
char Console::GetChar() char ch = incoming;
incoming = EOF; return ch;
Input : A character to be displayed Output : The character that is entered by the user will be displayed again
3 - void PutString(char***c**);
void print (char* c) PutString(c);
int main() print("Hello World. Here Come The Robots. This was a pleasure to work with you."); return 0;
Input : A string to be displayed Output : The string
4- void PutInt(int **num**);
Testcase int main() int n; GetInt(&n); PutInt(n); Halt();
Input : An integer to be displayed Output: The integer
5 - void GetInt(int ***num**);
Testcase : int main() int n; GetInt(&n); PutInt(n); return 0;
Input : Read an integer from the console

5.2 Thread System Call Test

1 - UserThreadCreate: Test Script
Input: Output: Conclusion:

6 Implementation Issue

6.1 Synchconsole class

1 Synchronization Issue: Based on the given structure, we have one primitive console class which can only support read(getchar) and write(putchar) one character in asynchronization manner. In that case, We developed a new class named Synchconsole to work in synchronization way and support several new methods like synputchar and syngetchar which is the synchronization version of putchar and getchar in console class. We solve this issue by using semaphore writeDone and readAvail to synchronize read and write operation on the simulated console environment.

2 String read/write Issue: In order to string, we have developed two new methods called synchputstring to write string onto the console and synchreadstring to read string from the console.

For SynchReadString(char *buffer,int n): first, we get the two argument value from register. buffer is a char pointer, we need to get the physical address value before invoking SynchReadString() because it only works on the kernel level. We get the physical address of buffer by accessing machine-¿mainMemory[].

For SynchWriteString(char *buffer): In order to get the physical address of string buffer in kernel level, we need to develop a new internal rountine called as CopyStringFromMachine(int from, char *to, int n) which used to copy at most n-length characters from the content of what virtual addr 'from' points to to

a place in phy addr pointed by a char pointer 'to'. CopyStringFromMachine will use a internal routine named machine-¿ReadMem to read the content of virtual addr into phy addr. /**need more since CopyStringFromMachine is a important function**/

3 Int read/write Issue: We use two functions in c library which are snprintf and sscanf. sscanf will format input content into integar type and snprintf will format integar type into character type. By using those two functions, we can use snprintf to transfer char into int when we use SynchPutInt to write integar value to the simulated standard output by provoking synchPutChar ; we can also use SynchGetInt to read character value from simulated standard output by provoking synchGetChar and then we use sscanf to format char into a int variable.

6.2 UserThread

1 Create:

2 Exit:

6.3 Implementing synchconsole and syscalls

Note: In case of "GetChar", when we get a character from console then only we can call Getchar(): If we do not test this condition and call the 'GetChar' then 'GetChar' could return 'EOF' but there is already a character which could not be fetched. This condition applies in case of 'PutChar' too. To print a 'char', we have an internal buffer. In this, Before printing another 'char', we will have to wait.

If we remove the Halt function, make fails because Main() needs a return value. If we make it return 0 instead of Halt, without changing anything else, running the program (make works now) produces a Unexpected user mode exception. Our solution was to create an exit system call and to store the return parameter in a register.

From the buffer, we are trying to read as much as we can(defined in the Size), for the rest, we truncate them.

Part VI -¿ If we remove the call to Halt() at the end of the main function of putchar, then when we 'make' for compilation then we get an error which states that "control reaches end of non-void function".

6.4 Synchronization for read and write operation in Console

6.5 Multithreading

7 Group Work Ogranization

7.1 Step 1 - 4

8 Conclusion