TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG MÁY TÍNH - VIỄN THÔNG

BÀI BÁO CÁO

Project 2 – Hệ điều hành Nachos

Môn học: Hệ điều hành

Lớp: 22CLC06

Giáo viên hướng dẫn: Ths. Lê Viết Long

Sinh viên thực hiện:

- **22127022 - Võ Hoàng Anh**
- **22127154 - Nguyễn Gia Huy**
- **22127210 - Phạm Anh Khôi**
- **22127413 - Phạm Hoàng Tiên**

Mục lục

I. Thông tin nhóm

- Môn: Hệ điều hành
- Lớp: 22CLC06
- Project 2 - Hệ điều hành Nachos
- Thành viên nhóm
+ 22127022 - Võ Hoàng Anh
+ 22127154 - Nguyễn Gia Huy
+ 22127210 - Phạm Anh Khôi
+ 22127413 - Phạm Hoàng Tiên

II. Bảng phân công công việc

MSSV	Họ và tên	Công việc
22127022	Võ Hoàng Anh	`ReadInt, PrintInt, ReadChar, PrintChar`
22127154	Nguyễn Gia Huy	`CompareFloat, FreeFloat, FloatToString`, chương trình minh họa, viết báo cáo
22127210	Phạm Anh Khôi	`ReadFloat, PrintFloat, ReadString, PrintString`
22127413	Phạm Hoàng Tiên	`Create, Open, Close, Read, Write`

III. Đánh giá mức độ hoàn thành

STT	Tên công việc	Mức độ hoàn thành
1	`ReadInt, PrintInt`	Hoàn thành
2	`ReadFloat, PrintFloat`	Hoàn thành
3	`ReadChar, PrintChar`	Hoàn thành
4	`ReadString, PrintString`	Hoàn thành
5	`Create, Open, Close`	Hoàn thành
6	`Read, Write`	Hoàn thành
7	`help, ascii`	Hoàn thành
8	`quicksort, mergesort`	Hoàn thành

IV. Các bước thực hiện

Bảng dưới đây là danh sách các file chính mà nhóm đã làm việc trong project lần này

STT	Tên file	Ý nghĩa	Mật độ
1	`syscall.h`	Khai báo các `System call(SC)`	10%
2	`exception.cc`	Viết các xử lí ngoại lệ cho `SC`	70%
3	`start.*`	Khai báo `SC` dưới dạng `Assembly`	5%
4	`openfile.*, filesys.*`	Hỗ trợ các thao tác với file	5%
5	`./test/*`	Viết các chương trình người dùng	10%

A. Cài đặt Nachos trên Ubuntu

- Yêu cầu: Ubuntu phiên bản 14.04 32 bit
- Link download: https://drive.google.com/uc?id=0B9icmOO7kaOncHpYTS1sN2Z0MkU&export=download

Các bước tiến hành: Download nguyên gói Nachos đã cấu hình kèm cross-compiler-mips (download tập tin nachos.zip trong thư mục tài liệu hướng dẫn)

Tạo thư mục HDH ở Desktop để tiện làm việc

+ Bước 1: Cài đặt gcc và g++
    + `sudo apt-get install gcc`
    + `sudo apt-get install g++`
+ Bước 2: Giải nén nachos ở Desktop/HDH
+ Bước 3: Sửa Makefile (ở thư mục nachos 3.4/code): Sửa dòng `MAKE = gmake ->` `MAKE = make`
+ Bước 4: Truy cập vào thư mục code và biên dịch nachos bằng lệnh `make`
+ Bước 5: Chạy thử chương trình: vào thư mục `threads`, chạy lệnh nachos: `./nachos`

Kết quả như sau là thành công:

```
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 10, idle 0, system 10, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
```

Một số lưu ý:

+ Có thể sử dụng lệnh `./userprog/nachos -rs 1023 -x ./test/halt` để chạy lệnh `halt` bằng chương trình `nachos`. Ý nghĩa cụ thể của các tham số trong câu lệnh sau:
  - `./userprog/nachos`: chạy chương trình nachos trong thư mục ./code/userprog
  - `-rs`: Cờ Random seed
  - `1023`: Giá trị của cờ random seed (Giá trị này có thể là bất kì giá trị nào)
  - `-x`: Cờ thông báo chạy 1 chương trình người dùng
  - `./test/halt`: Giá trị của cờ là chương trình halt trong thư mục `./code/test`
+ Khi biên dịch bằng make sẽ tạo ra nhiều chương trình nachos ở các thư mục, một số `nachos` ở các thư mục sẽ giống nhau nhưng ta thống nhất sử dụng chương trình `nachos` ở thư mục `./userprog`. Chạy một chương trình người dùng thông qua lệnh (tại thư mục ./code)

```
./userprog/nachos -x ./test/[Tên chương trình]
```

B. Đọc hiểu các tập tin trong đồ án

+ **progtest.cc** kiểm tra các thủ tục để chạy chương trình người dùng
+ **syscall.h** system call interface: các thủ tục ở kernel mà chương trình người dùng có thể gọi
+ **exception.cc** xử lý system call và các exception khác ở mức user, ví dụ như lỗi trang, trong phần mã chúng tôi cung cấp, chỉ có 'halt' system call được viết
+ **bitmap.*** các hàm xử lý cho lớp bitmap (hữu ích cho việc lưu vết các ô nhớ vật lý) **filesys.h**
+ **openfile.h** định nghĩa các hàm trong hệ thống file nachos. Trong đồ án này chúng ta sử dụng lời gọi thao tác với file trực tiếp từ Linux, trong đồ án khác chúng ta sẽ triển khai hệ thống file trên ổ đĩa giả lập. (nếu kịp thời gian)
+ **translate.*** Phiên bản nachos chúng tôi gửi các bạn, chúng tôi giả sử mỗi địa chỉ ảo là cũng giống hệt như địa chỉ vật lý, điều này giới hạn chúng ta chỉ chạy 1 chương trình tại một thời điểm. Các bạn có thể viết lại phần này để cho phép nhiều chương trình chạy cùng lúc trong đồ án sau.
+ **machine.*** mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng: bộ nhớ chính, thanh ghi, v.v.
+ **mipssim.cc** mô phỏng tập lệnh của MIPS R2/3000 processor (**Chứa lệnh tăng PROGRAM COUNTER, tìm từ khóa Advanced**, dung trước khi trả giá trị về của các systemcall)

+ **console.*** mô phỏng thiết bị đầu cuối sử dụng UNIX files. Một thiết bị có đặc tính (i) đơn vị dữ liệu theo byte, (ii) đọc và ghi các bytes cùng một thời điểm, (iii) các bytes đến bất đồng bộ
+ **synchconsole.*** nhóm hàm cho việc quản lý nhập xuất I/O theo dòng trong Nachos.
+ **../test/*** Các chương trình C sẽ được biên dịch theo MIPS và chạy trong Nachos
+ **system.*** chứa khai báo và xử lí các biến toàn cục sử dụng trong Nachos

C. Hiểu thiết kế của hệ điều hành

Giống như các hệ điều hành hiện đại, hệ điều hành Nachos cũng chia thành 2 không gian bộ nhớ:

+ **Kernel - system space**: không gian hệ thống, chứa các thông tin về chương trình người dùng
+ **User space**: không gian người dùng, chứa các thông tin về biến, hàm, bộ nhớ, chương trình của hệ thống

Việc chuyển quyền điều khiển từ **user mode** thành **system mode** được thực hiện thông qua `system calls`, software interrupt/trap.

Vì vùng nhớ *user space* không có ý nghĩa với *kernel*, chúng ta cần chuyển nội dung từ *user space* vào *kernel* sao cho kernel có thể xử lí được dữ liệu này. Khi trả thông tin từ system về *user space*, thì các giá trị được đặt trong các thanh ghi của CPU

Dưới đây là một số ví dụ thực tế minh họa cho các lí thuyết ở trên bằng mã nguồn

```
/// @brief Read a string from the console input
/// @param buffer Buffer to store the string (Stored in register 4)
/// @param size Size of the buffer (Length of the string) (Stored in register 5)
void ReadString(char *buffer, int size);
```

```
/// @brief Handle system call ReadString from user program
void Handle_SC_ReadString()
{
    int virtAddr;        // Virtual address of input string from user space
    int length;          // Length of input string
    char *buffer = NULL; // Buffer to store input string

    virtAddr = machine->ReadRegister(4); // Read virtual address of input string PARAMETER from register 4
    length = machine->ReadRegister(5);   // Read maximum length of input string PARAMETER from register 5

    buffer = User2System(virtAddr, length); // Copy buffer from User memory space to System memory space

    gSynchConsole->Read(buffer, length); // Use SynchConsole to read buffer from console

    System2User(virtAddr, length, buffer); // Copy buffer from System memory space to User memory space

    delete buffer;
    return IncreasePC();
}
```

Ví dụ trên minh họa về `System call ReadString` với 2 tham số đầu vào là `buffer` và `size` dùng để đọc một chuỗi kí tự từ `Console`. Ở chương trình người dùng, khi gọi hàm (từ bây giờ sẽ dùng từ `Syscall` , là các hàm được khai báo trong `syscall.h`). Tham số từ chương trình người dùng truyền vào `Syscall` sẽ được lưu trong các thanh ghi, cụ thể trong `Syscall` trên, tham số `buffer` được lưu trong thanh ghi thứ 4 và tham số `size` được lưu trong thanh ghi thứ 5.
Ở `Kernel space`, hàm `Handle_SC_ReadString` xử lí `Syscall ReadString` sẽ sử dụng các biến cục bộ của Kernel với dữ liệu được đọc từ `User space` thông qua `machine->ReadRegister`. Như ở ví dụ trên thì giá trị của `buffer` sẽ được truyền vào tham số `virtAddr` thông qua phương thức đọc thanh ghi thứ 4 của `machine`, tương tự như vậy với biến `length`.

Tuy nhiên, giá trị địa chỉ truyền vào `buffer` của `User space` (lúc này được lưu trong biến `virtAddr`) chỉ là địa chỉ ảo của `User space`. Ta cần chuyển dữ liệu từ `User space` sang `Kernel` thông qua hàm `User2System`. Tại đây, ta có được dữ liệu `buffer` của `Kernel` có giá trị bằng với `buffer` của `User Space` (nhưng khác về địa chỉ).
Sau đó ta đọc dữ liệu chuỗi nhập vào thông qua dòng tiếp theo `gSynchConsole->Read(buffer, length)`, phần sau sẽ nói rõ hơn về `class SynchConsole` và các phương thức xử lí của nó

Sau khi đọc dữ liệu vào `buffer` của `Kernel` xong, ta thực hiện sao chép giá trị của `buffer` ở `Kernel` vào `buffer` của `User space` (lúc này là biến `virtAddr`). Sau khi sao chép xong, giá trị của `buffer` ở `Kernel` và `User space` là như nhau (tuy nhiên vẫn khác địa chỉ)

Cuối cùng ta xóa vùng nhớ đã cấp phát ở `Kernel` trong biến `buffer` và tăng giá trị `Program Counter` (điều này là bắt buộc với mỗi hàm xử lí `Syscall`)

D. Sử dụng `Class SynchConsole`

1. Tại sao cần dùng `Class SynchConsole` ?

`Class SynchConsole` hỗ trợ đọc và ghi dữ liệu từ `Kernel` vào `User space`. Cụ thể, lớp này hỗ trợ 2 phương thức `Read` để đọc dữ liệu từ `Terminal` (`User space`) vào `Kernel buffer` và phương thức `Write` để ghi dữ liệu từ `Kernel buffer` vào `Terminal`. Phần khai báo của 2 phương thức được mô tả chi tiết như đoạn mã bên dưới.

```
/// @brief Read a buffer from the console input device and store it in the kernel buffer
/// @param into Buffer to store the input
/// @param numBytes Number of bytes to read
/// @return Number of bytes actually read
int Read(char *into, int numBytes);

/// @brief Write a buffer from the kernel to the console output device
/// @param from Kernel buffer to write
/// @param numBytes Number of bytes to write
/// @return Number of bytes actually written
int Write(char *from, int numBytes);
```

Tóm lại, nếu các `Register` trong `machine` được dùng để đọc và ghi giá trị tham số, giá trị trả về của `Syscall` thì `SynchConsole` dùng để đọc dữ liệu từ `Userspace` và ghi dữ liệu vào `User space`

2. Cách thêm `Class SynchConsole` vào `Nachos`

+ **Bước 1**: Chuẩn bị 2 file `synchcons.h` và `synchcons.cc` đã được cung cấp trong đồ án
+ **Bước 2**: Sao chép 2 file trên vào thư mục `/code/threads`
+ **Bước 3**: Mở file `Makefile.common`, thêm vào cuối các mục `USERPROG_H`, `USERPROG_C`, `USERPROG_O` các đường dẫn như hình bên dưới

```
USERPROG_H = ../userprog/addrspace.h\
    ../userprog/bitmap.h\
    ../filesys/filesys.h\
    ../filesys/openfile.h\
    ../machine/console.h\
    ../machine/machine.h\
    ../machine/mipssim.h\
    ../machine/translate.h\
    ../threads/synchcons.h

USERPROG_C = ../userprog/addrspace.cc\
    ../userprog/bitmap.cc\
    ../userprog/exception.cc\
    ../userprog/progtest.cc\
    ../machine/console.cc\
    ../machine/machine.cc\
    ../machine/mipssim.cc\
    ../machine/translate.cc\
    ../threads/synchcons.cc

USERPROG_O = addrspace.o bitmap.o exception.o progtest.o console.o machine.o \
    mipssim.o translate.o synchcons.o      You, yesterday • update
```

+ **Bước 4**: Trong `/code/threads`, mở `system.h`, khai báo biến `extern` để có thể sử dụng trong toàn bộ mã nguồn trong phần `USER_PROGRAM`

```
#ifdef USER_PROGRAM
#include "machine.h"
#include "synchcons.h"
#include "synch.h"
extern Machine *machine;
extern SynchConsole *gSynchConsole;

#endif
```

+ **Bước 5**: Tiếp tục mở `system.cc`, tại các mục `USER_PROGRAM`, khai báo biến, cấp phát bộ nhớ khi bắt đầu vào thu hồi bộ nhớ đã cấp phát cho biến khi kết thúc chương trình. Tham khảo hình minh họa bên dưới

```
#ifdef USER_PROGRAM           // requires either FILESYS or FILESYS_STUB
Machine *machine;             // user program memory and registers
SynchConsole *gSynchConsole; // the synchronized console
#endif
```

```
#ifdef USER_PROGRAM
    machine = new Machine(debugUserProg); // this must come first
    gSynchConsole = new SynchConsole();
#endif
```

```
#ifdef USER_PROGRAM
    delete machine;
    delete gSynchConsole;
#endif
```

+ **Bước 6**: Sử dụng biến `gSynchConsole` để xử lí các `Syscall` trong `exception.cc`

E. Cài đặt các `Syscall`

1. `Syscall Read - Print` cho `Int - Float - Char - String`

Vì các chương trình người dùng chỉ có thể dùng các `Syscall` để tương tác với hệ điều hành nên cần phải viết các `Syscall` cho các thao tác nhập xuất các kiểu dữ liệu khác nhau (Chứ không thể `scanf`, `printf` đối với `C` hay `cin, cout` với `C++`). Sau đây là hướng dẫn chi tiết các bước để viết một `Syscall`, để đơn giản trong quá trình xử lí logic, ta sẽ minh họa xử lí `Syscall ReadChar` và `PrintChar`

+ Bước 1: Mở `syscall.h`, định nghĩa giá trị cho `Syscall` (kí hiệu là `SC_` + tên `Syscall`). Hình bên dưới định nghĩa giá trị của `SC_ReadChar` là 45 và `SC_PrintChar` là 46

```
13  #ifndef SYSCALLS_H
14  #define SYSCALLS_H
15
16  #include "copyright.h"
17
18  /* system call codes -- used by the stubs to tell the kernel which system call
19   * is being asked for
20   */
21  #define SC_Halt 0
22  #define SC_Exit 1
23  #define SC_Exec 2
24  #define SC_Join 3
25  #define SC_CreateFile 4
26  #define SC_Open 5
27  #define SC_Read 6
28  #define SC_Write 7
29  #define SC_Close 8
30  #define SC_Fork 9
31  #define SC_Yield 10
32
33  #define SC_ReadInt 41
34  #define SC_PrintInt 42
35  #define SC_ReadFloat 43
36  #define SC_PrintFloat 44
37  #define SC_ReadChar 45      You, 2 days ago • [code]: Write syscall for I/O Int, Char, Strin
38  #define SC_PrintChar 46
39  #define SC_ReadString 47
40  #define SC_PrintString 48
41
```

+ Bước 2: Trong `syscall.h`, sau khi định nghĩa, ta khai báo các `Syscall` dưới dạng các hàm như hình minh họa dưới đây

```
72
73  /// @brief Read a character from the console input and write it to register 2
74  /// @return Character read from the console input
75  char ReadChar();
76
77  /// @brief Write a character to the console output
78  /// @param character Character is stored in register 4
79  void PrintChar(char character);      You, 2 days ago • [code]: Write syscall for
80
81  /// @brief Read a string from the console input
82  /// @param buffer Buffer to store the string (Stored in register 4)
83  /// @param size Size of the buffer (Length of the string) (Stored in register 5)
84  void ReadString(char *buffer, int size);
85
86  /// @brief Write a string to the console output
87  /// @param buffer Buffer contains the string (Stored in register 4)
88  void PrintString(char *buffer);
89
```

+ Bước 3: Trong thư mục `/code/test`, mở `start.s` và `start.c`, thêm vào 2 file này đoạn mã như hình bên dưới

```
151
152         .globl ReadChar
153         .ent    ReadChar
154     ReadChar:
155         addiu $2,$0,SC_ReadChar
156         syscall
157         j   $31
158         .end ReadChar
159
160         .globl PrintChar
161         .ent    PrintChar
162     PrintChar:
163         addiu $2,$0,SC_PrintChar
164         syscall
165         j   $31
166         .end PrintChar
167
```

+ Bước 4: Trong thư mục `code/userprog`, mở `exception.cc`, viết mã nguồn xử lí `Syscall`. Hình bên dưới minh họa cấu trúc tổng thể (sử dụng cấu trúc `switch-case` của hàm `ExceptionHandler` để xử lí các `Exception`. Trong đó `SyscallException` xảy ra khi gọi `Syscall`

```
560  /// @brief Exception handler for user program system calls
561  /// @param which Type of exception
562  void ExceptionHandler(ExceptionType which)
563  {
564      int type = machine->ReadRegister(2); // Read system call code from register 2
565
566      switch (which)
567      {
568      case SyscallException: // System call exception
569          switch (type)
570          {
571          case SC_Halt:
572              return Handle_SC_Halt();
573          case SC_ReadInt:
574              return Handle_SC_ReadInt();
575          case SC_PrintInt:
576              return Handle_SC_PrintInt();
577          case SC_ReadFloat:
578              // SynchPrint("ReadFloat\n");
579              return Handle_SC_ReadFloat();
580          case SC_PrintFloat:
581              // SynchPrint("PrintFloat\n");
582              return Handle_SC_PrintFloat();
583          case SC_ReadChar:
584              return Handle_SC_ReadChar();
585          case SC_PrintChar:
586              return Handle_SC_PrintChar();      You, 2 days ago • [code]: Write sysca
587          case SC_ReadString:
588              return Handle_SC_ReadString();
589          case SC_PrintString:
590              return Handle_SC_PrintString();
591          case SC_CreateFile:
592              return Handle_SC_CreateFile();
593          case SC_Open:
594              return Handle_SC_Open();
595          case SC_Close:
596              return Handle_SC_Close();
597          case SC_Read:
598              return Handle_SC_Read();
599          case SC_Write:
600              return Handle_SC_Write();
601          default:
602              interrupt->Halt();
603              break;
604          }
605      case NoException:
606          return;
607      case PageFaultException:
608          printf("PageFaultException: No valid translation found\n");
609          interrupt->Halt();
610          break;
611      case ReadOnlyException:
612          printf("ReadOnlyException: Write attempted to page marked \"read-only\"\n");
613          interrupt->Halt();
614          break;
```

Về cấu trúc tổng thể là như vậy. Tuy nhiên, để chương trình chạy đúng, ta cần xử lí chính xác các `Syscall`. Dưới đây là hình minh họa mã nguồn và giải thích chi tiết mã nguồn của hàm `Handle_SC_ReadChar` để xử lí khi gọi `SC_ReadChar` và `Handle_SC_PrintChar` để xử lí khi gọi `SC_PrintChar`

a. SC_ReadChar

```
143
144  /// @brief Handle system call ReadChar from user program
145  void Handle_SC_ReadChar()
146  {
147      int maxBytes = 255;          // Maximum length of buffer
148      char *buffer = new char[255]; // Buffer to store data
149      int numBytes = 0;            // The number of bytes read from console
150      char c = 0;                  // Character to store the result
151
152  >   if (buffer == NULL) // Check if buffer is NULL (not enough memory in system)···
156      else // Buffer is not NULL
157      {
158          numBytes = gSynchConsole->Read(buffer, maxBytes); // Read buffer from console and return the number of bytes read
159
160  >       if (numBytes > 1) // Check if the number of bytes read is greater than 1···
165  >       else if (numBytes == 0) // Check if the number of bytes read is 0: empty character···
170          else // If the number of bytes read is 1, return the character to register 2
171          {
172              c = *buffer;
173          }
174      }
175      machine->WriteRegister(2, c); // Write the character to register 2
176      delete buffer;
177      return IncreasePC();
178  }
179
```

Chú thích chi tiết của hàm xử lí đã được comment ở đoạn `code` trên. Có một số điểm cần lưu ý như sau:

+ Sử dụng phương thức `Read` của `gSynchConsole` để đọc dữ liệu từ `User space`
+ Sau khi xử lí xong về mặt logic, ghi kết quả (kí tự được nhập) vào `Register[2]`, vì thanh ghi thứ 2 lưu giá trị trả về của `Syscall`. Phải ghi vào thì giá trị `Syscall` (giá trị của hàm `ReadChar` mới trả về giá trị để lưu vào biến khi sử dụng `Syscall` (hàm) trong chương trình người dùng
+ Cần tăng `Program Counter` sau mỗi `Syscall` để chương trình hoạt động đúng. Dưới đây là minh họa về hàm tăng `Program Counter`

```
56  /// @brief Increase Program Counter (needed for each system call) (4 bytes for each instruction)
57  void IncreasePC()
58  {
59      int counter = machine->ReadRegister(PCReg);     // Read current Program Counter
60      machine->WriteRegister(PrevPCReg, counter);     // Write current Program Counter to Previous Program Counter
61      counter = machine->ReadRegister(NextPCReg);     // Read Next Program Counter
62      machine->WriteRegister(PCReg, counter);         // Write Next Program Counter to Program Counter
63      machine->WriteRegister(NextPCReg, counter + 4); // Write Next Program Counter + 4 to Next Program Counter
64  }
```

b. SC_PrintChar

```
/// @brief Handle system call PrintChar from user program
void Handle_SC_PrintChar()
{
    char c = (char)machine->ReadRegister(4); // Read character parameter from register 4

    if (c != 0) // Check if the character is not null
    {
        // SynchPrint("Character that you entered: ");
        gSynchConsole->Write(&c, 1); // Write the character to console
    }

    return IncreasePC();
}
```

Tương tự như `SC_ReadChar`, chú thích chi tiết đã được `comment code` trên mã nguồn. Một số lưu ý:

+ Đọc giá trị tham số đầu vào của `Syscall void PrintChar(char c)` ở thanh ghi thứ 4 và lưu vào biến cục bộ của hàm
+ Sử dụng phương thức `Write` của `gSynchConsole` để in kí tự đọc được ra màn hình
+ Tăng `Program Counter` thông qua hàm `IncreasePC`

c. Các `Syscall` còn lại

Các `Syscall` nhập xuất cho các kiểu dữ liệu còn lại trong đồ án này cũng như các kiểu dữ liệu khác được cài đặt tương tự về mặt cấu trúc, chỉ khác phần xử lí logic tương ứng với từng kiểu dữ liệu khác nhau

+ `SC_ReadInt`
+ `SC_PrintInt`
+ `SC_ReadFloat`
+ `SC_PrintFloat`
+ `SC_ReadString`
+ `SC_PrintString`

Lưu ý:

+ Vì thanh ghi trong `machine` chỉ lưu được giá trị số nguyên nên `SC_ReadFloat` và `SC_PrintFloat` cần thao tác với `float*` để đảm bảo tính đúng đắn.
+ Vì chương trình người dùng không hỗ trợ toán tử giải tham chiếu, vì vậy khi muốn thực hiện các thao tác khác với số thực, ta cần viết các `System call` cho chúng. Ví dụ như `CompareFloat, FreeFloat, FloatToString`

2. `Syscall Create, Open, Close, Read, Write` cho File

Đối với các `Syscall` liên quan để xử lí `File`, chúng ta cần điều chỉnh lại các file `openfile.*` và `filesys.*` trong thư mục `/code/filesys`.

a. `openfile.*`

`Openfile.h` định nghĩa `class OpenFile` và các phương thức xử lí file trong hệ thống file `nachos` như: `Read`, `Write`, `Seek`, `Length`, `GetCurrentPos` để hỗ trợ cho phần cài đặt

Với đoạn mã nguồn được cho trước trong đồ án này, ta cần thêm vào `class` thuộc tính `type` để biểu diễn trạng thái của file (`openmode`) và cài đặt thêm `Constructor` có tham số `type` như hình minh họa bên dưới

```
108
109   #define READ_WRITE 0
110   #define READ_ONLY 1
111   #define STDIN 2
112   #define STDOUT 3
      You, 57 seconds ago | 1 author (You)
113   class OpenFile
114   {
115   public:
116       int type; // Open mode
117       // Open a file whose header is located at "sector" on the disk
118       OpenFile(int sector);
119       // Open a file with a specific type
120       OpenFile(int sector, int type);
121
```

`Type` có thể mang 1 trong 4 giá trị được định nghĩa như trên hình:

+ `READ_WRITE`: file cho phép đọc và ghi
+ `READ_ONLY`: file chỉ cho phép đọc
+ `STDIN`: thiết bị nhập chuẩn (bàn phím)
+ `STDOUT`: thiết bị xuất chuẩn (màn hình)

b. `filesys.*`

`Fileys.h` định nghĩa `class FileSystem` để quản lí các tập tin trong hệ thống file `nachos`, hỗ trợ một số phương thức như `Open` để mở file, `Remove` để đóng file, `Print` để in danh sách file hiện hành, …

Với đoạn mã được cung cấp trong `filesys.*` của đồ án lần này, ta cần thêm vào `class FileSystem` một số thuộc tính và phương thức để quản lí và xử lí bảng file. Hình bên dưới là ví dụ minh họa về cách cài đặt bảng file đơn giản

```
126  class FileSystem
127  {
128  public:
129      OpenFile **file_table; // A table to store all the file
130      int index;             // Index of the file table
131
132      /// @brief Open the file with the given name and open mode
133      /// @param name Name of the file in ./code directory
134      /// @param type Type of the file (0->3)
135      /// @return OpenFile pointer of the opened file
136      OpenFile *Open(char *name, int type);
137
```

Thuộc tính `file_table` là một con trỏ `OpenFile` cấp 2 để quản lí danh sách các file trong hệ thống. Mỗi khi một file được mở, chương trình sẽ cấp phát bộ nhớ trên 1 vị trí của `file_table` để quản lí file đó. Khi đóng file, chương trình sẽ thu hồi bộ nhớ từ vùng đã được cấp phát cho file đó.

c. Syscall SC_Open

`SC_Open` dùng để mở một file trên hệ thống file `nachos`. Ví dụ về một cách cài đặt hàm xử lí `SC_Open` được minh họa chi tiết trong hình dưới đây

+ Trong `syscall.h`

```
115
116  /// @brief Open the file with the given name and open mode
117  /// @param name Name of the file in ./code directory (Stored in register 4)
118  /// @param type Type of the file (0: Read/Write, 1: Read only, 2: Console Inpu
119  /// @return OpenFileId of the opened file (a slot in file table)
120  /// @note + Return -1 if the file is not found or the file is already opened
121  /// @note + Return -2 if unknown error occurs
122  OpenFileId Open(char *name, int type);
123
```

+ Trong `exception.cc`

```
372
373 /// @brief Handle system call Open from user program (Open file)
374 void Handle_SC_Open()
375 {
376     int virtAddr = machine->ReadRegister(4); // Virtual address of filename PARAMETER from register 4
377     int type = machine->ReadRegister(5);     // Type (Open mode) of the file PARAMETER from register 5
378     char *filename;                          // Buffer to store filename
379     int result = -2;                         // Result of the function
380
381     filename = User2System(virtAddr, MaxFileLength); // Copy buffer from User memory space to System memory space
382
383     int allocatedSlot = fileSystem->GetAllocatedSlot(); // Find free slot in file system
384
385     if (allocatedSlot != -1) // If there is free slot
386     {
387         if (type == READ_WRITE || type == READ_ONLY) // Handle ReadOnly and ReadWrite file cases
388         {
389             fileSystem->file_table[allocatedSlot] = fileSystem->Open(filename, type); // Open file with filename and type
390
391             if (fileSystem->file_table[allocatedSlot] != NULL)
392             {
393                 result = allocatedSlot; // Success, return free slot for normal file
394             }
395             else // Open file failed
396             {
397                 result = -1; // Failed to open file, return -1
398             }
399         }
400         else if (type == STDIN) // ConsoleInput: stdin
401         {
402             result = 0; // Success, return 0 for STDIN
403         }
404         else // ConsoleOutput: stdout
405         {
406             result = 1; // Success, return 1 for STDOUT
407         }
408     }
409 >   else // No free slot···
414
415     if (filename != NULL)
416         delete[] filename;
417
418     machine->WriteRegister(2, result); // Write result to register 2
419     return IncreasePC();
420 }
421
```

Một số lưu ý trong đoạn mã trên:

+ `fileSystem` là biến `extern FileSystem` được khai báo trong `system.h` ở phần `FILESYS_NEEDED`
+ Phương thức `GetAllocatedSlot()` trả về một vị trí trống trên bảng file
+ Phương thức `Open()` dùng để mở file và gán nó vào bảng file
+ Với mỗi giá trị của `type` sẽ ứng với mỗi cách xử lí khác nhau
+ Kết quả trả về của `Syscall` là vị trí của file trên bảng file

d. Các `Syscall` còn lại

Các `Syscall` cho `File` còn lại được cài đặt về mặt cấu trúc tương tự như `SC_Open`, chỉ khác về xử lí logic

+ `SC_CreateFile`: tạo file mới
+ `SC_Close`: đóng file (thu hồi vùng nhớ đã cấp phát trên `file_table`)
+ `SC_Read`: đọc dữ liệu từ file
+ `SC_Write`: ghi dữ liệu vào file

Chi tiết cách cài đặt và xử lí logic đã được viết và comment chi tiết trong mã nguồn (tham khảo `syscall.h` và `exception.cc`). Dưới đây là một số **lưu ý** trong quá trình cài đặt:

+ Đối với tên file / đường dẫn file được truyền vào tham số của `Syscall`, mặc định thư mục hiện hành của file đó là `./code`. Ví dụ nếu ta truyền vào `Syscall Create` tham số `filename = quicksort.txt` thì sẽ tạo file `quicksort.txt` tại thư mục `./code`. Tương tự như vậy với các `File Syscall` còn lại
+ Dùng các hàm `User2System` và `System2User` để chuyển đổi vùng nhớ qua lại giữa `Kernel` và `User space`
+ Xử lí logic đúng với từng `openmode` (`READ_WRITE, READ_ONLY, STDIN, STDOUT`) của file. Ví dụ với `Syscall Read` cho file `stdout` thì báo lỗi, hoặc `Write` cho file `stdout` thì cần dùng `gSynchConsole` để xuất dữ liệu ra màn hình.
+ Tăng `Program Counter` cuối mỗi `Syscall`

A. Viết chương trình người dùng

Trong phần này ta sẽ viết các chương trình người dùng - `User Program` sử dụng các `Syscall` của hệ thống. Để đơn giản hơn về mặt xử lí logic và tập trung vào cấu trúc tổng quát, ta viết chương trình `char_io_test` để xử lí nhập xuất 1 kí tự từ người dùng. Sau đây là chi tiết các bước thực hiện:

+ **Bước 1**: Tại thư mục `/code/test`, tạo file `char_io_test.c` . Hình bên dưới là minh họa đơn giản về chương trình này

```
You, 4 hours ago | 1 author (You)
#include "syscall.h"

#define NULL 0
#define BACKSPACE 8
#define ESC 27
#define SPACE 32
#define DELETE 127
#define TAB 9

int main()
{
    char c;
    PrintString("*====================================*\n");
    PrintString("| Welcome to the Char I/O Test Suite |\n");
    PrintString("*====================================*\n");

    PrintString("Enter a character: ");
    c = ReadChar();
    PrintString("You entered: ");
    switch (c)
    {
    case NULL:
        PrintString("NULL\n");
        break;
    case ESC:
        PrintString("ESC\n");
        break;
    case SPACE:
        PrintString("SPACE\n");
        break;
    case TAB:
        PrintString("TAB\n");
        break;
    default:
        PrintChar(c);
        break;
    }
    PrintChar('\n');
}       You, 2 days ago • [code]: Write syscall for I/O Int, Char, String
```

Chương trình sẽ cho phép người dùng nhập vào 1 kí tự và in ra kí tự vừa nhập, xử lí lỗi nếu nhập nhiều hơn 1 kí tự. Chương trình sử dụng `Syscall ReadChar` để đọc 1 kí tự từ thiết bị nhập của người dùng và `PrintChar` để in ra kí tự ra màn hình người dùng

Lưu ý: Khi viết các chương trình người dùng, các biến sử dụng trong chương trình tất cả nên được khai báo trước các lệnh xử lí để tránh gặp lỗi trong quá trình biên dịch. Ví dụ như trong `quicksort.c`, tất cả các biến liên quan đến mảng số nguyên, thuật toán `quickshort` và xử lí `file` được khai báo trước khi xử lí logic như hình minh họa bên dưới

```c
#include "syscall.h"

#define MAX_SIZE (100)

int main()
{
    int a[MAX_SIZE]; // Array to be sorted
    int n;           // Number of elements in the array
    int order;       // Order of sorting

    /* Quicksort */

    int stack[MAX_SIZE]; // Stack for storing left and right indices of sub arrays
    int top;             // Top of the stack
    int left;            // Left index of the sub array to be sorted
    int right;           // Right index of the sub array to be sorted
    int pivotIndex;      // Index of the pivot element
    int pivot;           // Pivot element
    int i, j;            // Loop variables
    int temp;            // Temporary variable for swapping

    /* File handling */
    int openFileId; // File descriptor for the file to be written
```

+ **Bước 2**: Mở `Makefile` trong thư mục `code/test` và thêm dữ liệu vào file như hình minh họa bên dưới để chương trình có thể hiểu và biên dịch

```
38  #----------------------------------------------------------------------------
39  all: halt char_io_test string_io_test int_io_test float_io_test help ascii quicksort mergesort
40  # ----------------------------------------------------------------------------
41  start.o: start.s ../userprog/syscall.h
42      $(CPP) $(CPPFLAGS) start.c > strt.s
43      $(AS) $(ASFLAGS) -o start.o strt.s
44      rm strt.s
45
46  # ----------------------------------------------------------------------------
47  char_io_test.o: char_io_test.c
48      $(CC) $(CFLAGS) -c char_io_test.c
49  char_io_test: char_io_test.o start.o
50      $(LD) $(LDFLAGS) start.o char_io_test.o -o char_io_test.coff
51      ../bin/coff2noff char_io_test.coff char_io_test
52  # ----------------------------------------------------------------------------
```

+ **Bước 3**: Chuyển hướng đến thư mục `code`, biên dịch bằng lệnh `make all`. Nếu biên dịch thành công, gõ lệnh sau để chạy chương trình `char_io_test`

  `./userprog/nachos -x ./test/char_io_test`

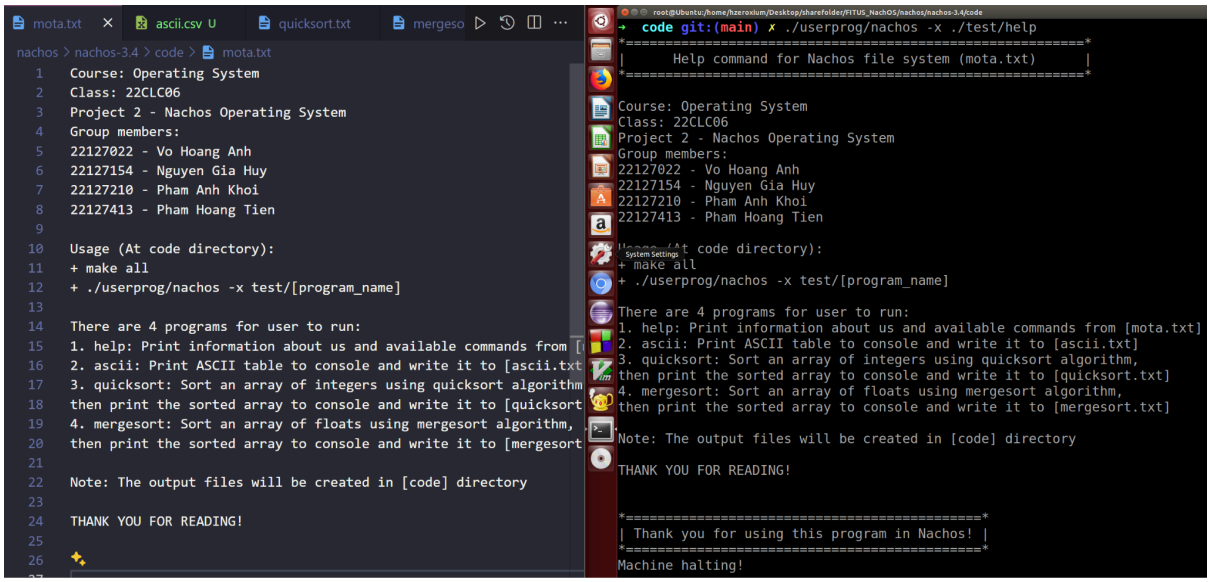
  Hình bên dưới là minh họa khi chương trình chạy thành công khi ta nhập `TAB`

```
→  code git:(main) ./userprog/nachos -x ./test/char_io_test
*=====================================*
| Welcome to the Char I/O Test Suite |
*=====================================*
Enter a character:
You entered: TAB

Machine halting!

Ticks: total 1300399758, idle 1300397890, system 1790, user 78
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 160
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
→  code git:(main)
```

V. Hình ảnh demo chương trình

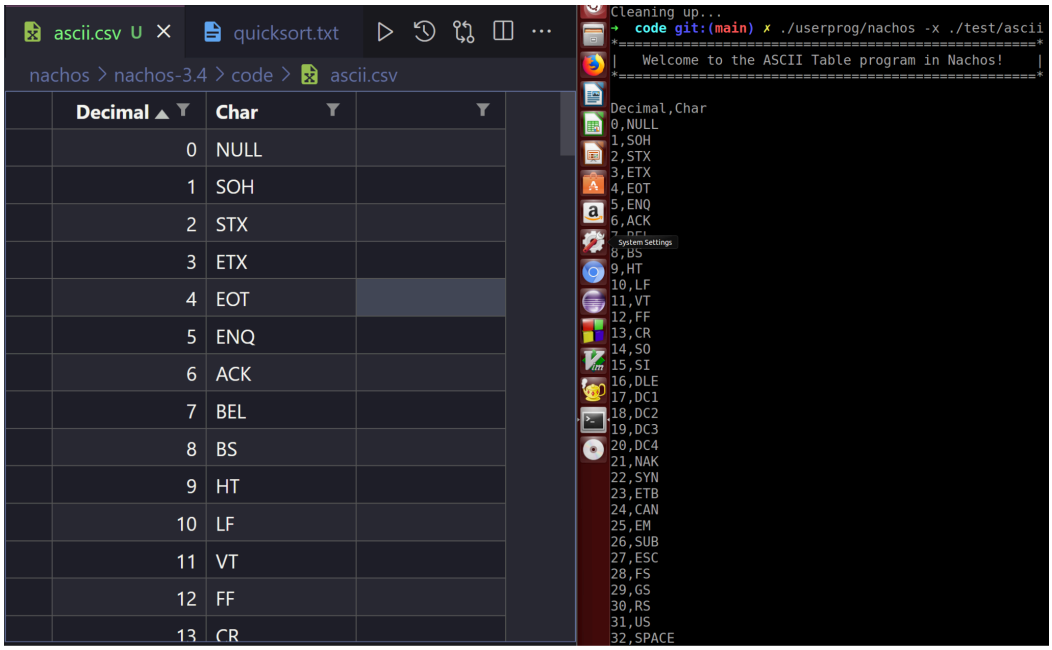
A. `help`

Chương trình `help` dùng để in ra màn hình các dòng giới thiệu cơ bản về nhóm. Nội dung mô tả được đọc từ tập tin `mota.txt`



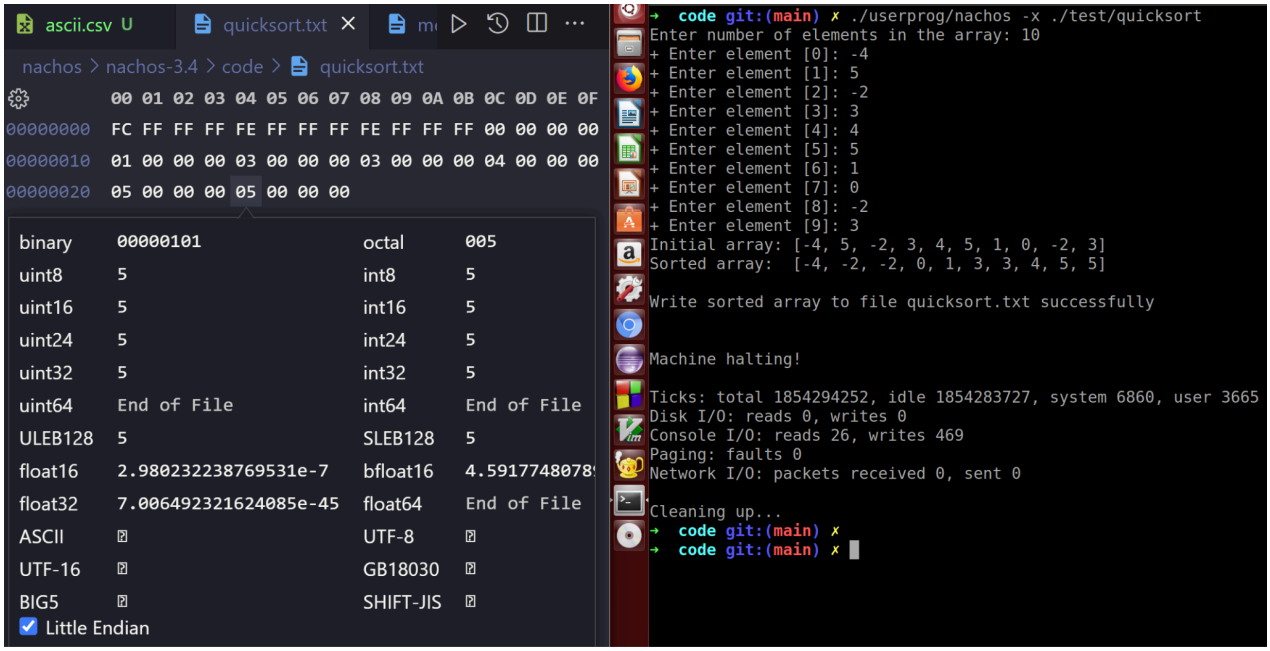
B. `ascii`

Chương trình `ascii` dùng để in ra màn hình bảng mã `ASCII` và ghi kết quả vào tập tin `ascii.csv`



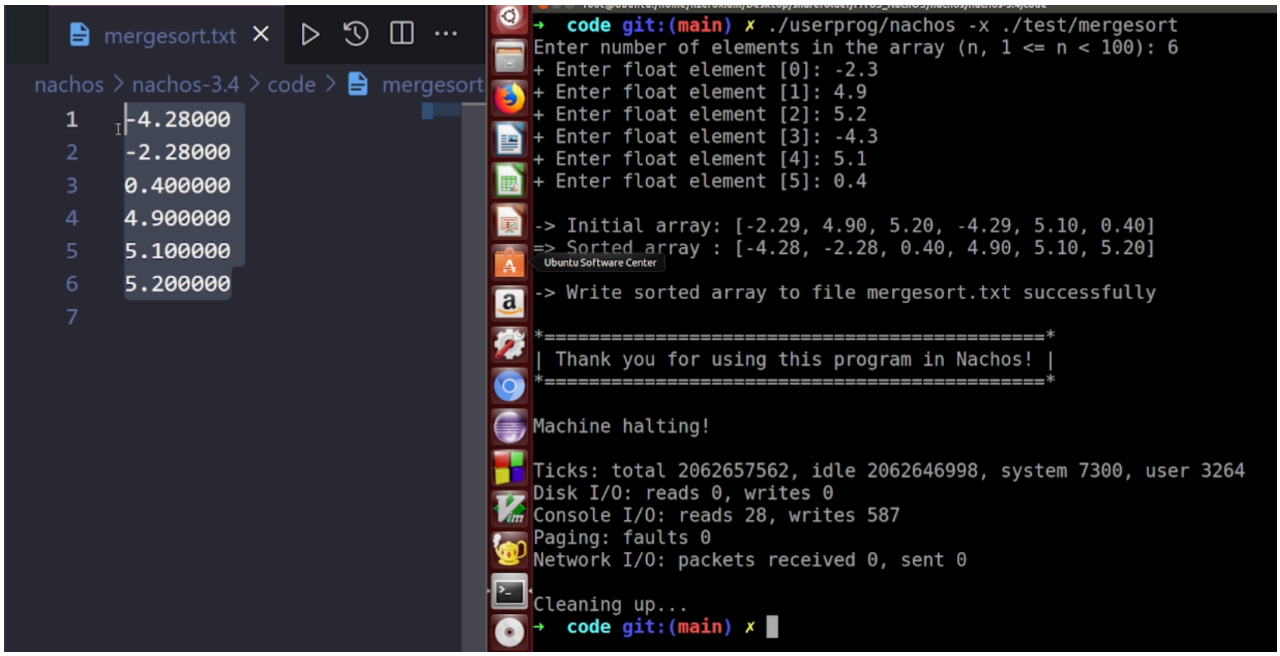
C. `quicksort`

Chương trình `quicksort` cho phép nhập vào n số nguyên từ bàn phím (`0 < n < 100`), sử dụng thuật toán **QuickSort** để sắp xếp và lưu kết quả vào tập tin `quicksort.txt`



D. `mergesort`

Chương trình `mergesort` cho phép nhập vào n số thực từ bàn phím (`0 < n < 100`), sử dụng thuật toán **MergeSort** để sắp xếp và lưu kết quả vào tập tin `mergesort.txt`



VI. Tài liệu tham khảo

[1]. Ths. Lê Viết Long, *Tài liệu Project 2 - 20240315*

[2]. Hào, H. C. [@hachihao1604]. (2021, October 12). FIT HCMUS | Hệ điều hành | Đồ án 1 NachOS. Youtube. https://www.youtube.com/watch?v=HOw6OdM1H-M

[3]. Nguyễn, T. C. [@thanhchungnguyen2618]. (n.d.). Lập trình nachos HCMUS. Youtube. Retrieved March 31, 2024, from https://www.youtube.com/playlist?list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO