

OPERATING SYSTEM LAB - 1

HỌC KÌ 212 (2021 - 2022)

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Hồ Đức Hưng, 2013381

Advisor:

PhD. Nguyễn Quang Hùng

Tóm tắt nội dung:

Trong bài báo cáo này em sẽ lần lượt trình bày các nội dung về phần:

3.1 QUESTION

3.2 BASIC COMMANDS

3.3 PROGRAMMING EXERCISES

Đính kèm sau là phần [Source Code](#) cho bài làm.

Academic year:

2021 - 2022

Key-words: makefile, vim, linux, ubuntu, command line, process, ...

Mục lục

1 QUESTION	1
1.1 What are the advantages of Makefile? Give examples	1
1.1.1 Advantages	1
1.1.2 Examples	2
1.2 In case of source code files located in different places, how can we write a Makefile?	3
1.3 What the output will be at LINE A? Explain your answer.	4
2 BASIC COMMANDS	4
3 PROGRAMMING EXERCISES	5
3.1 Problem 1	5
3.2 Problem 2	5
3.3 Problem 3	6
3.4 Problem 4	7

1. QUESTION

1.1. What are the advantages of Makefile? Give examples

1.1.1. Advantages

- Làm code ngắn gọn, dễ đọc, dễ sửa lỗi.
- Không cần phải biên dịch toàn bộ chương trình, bất cứ khi nào bạn thực hiện thay đổi đối với một chức năng hoặc một lớp. Makefile sẽ tự động biên dịch chỉ những tệp đã xảy ra thay đổi.
- Makefile được sử dụng rộng rãi để trình bày dự án một cách có hệ thống và hiệu quả hơn.

1.1.2. Examples

Chương trình tìm giai thừa theo cách truyền thống.

```
// Program to calculate factorial.
#include<bits/stdc++.h>
using namespace std;

// Function to find factorial
int factorial(int n)
{
    if (n == 1)
        return 1;
    // Recursive Function to find
    // factorial
    return n * factorial(n - 1);
}
// Function to print
void print()
{
    cout << "makefile" << endl;
}
// Driver code
int main()
{
    int fact = 5;
    cout << factorial(5) << endl;
    print();
    return 0;
}
```

Với đoạn chương trình trên nếu trong một dự án lớn, ta khá khó khăn khi quản lý các class. Vì vậy, ta chia ra các file class nhỏ hơn để dễ quản lý code hơn.

```
// file main.cpp
#include <bits/stdc++.h>
// Note function.h which has all functions
// definitions has been included
#include "function.h"

using namespace std;

// Main program
int main()
{
    int num3 = 5;
    cout << factorial(num3) << endl;
    print();
}
```

```
// factorial.cpp
#include <bits/stdc++.h>

// Definition of factorial function
// is present in function.h file
#include "function.h"
using namespace std;

// Recursive factorial program
```

```
int factorial(int n)
{
    if (n == 1)
        return 1;
    return n * factorial(n - 1);
}
```

```
// function.h
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

void print();
int factorial(int);

#endif
```

Ta có thể chỉnh sửa và quản lý các class trong từng file một cách dễ dàng, và biên dịch nó bằng Makefile.

```
// Open Terminal and type commands:
g++ -c main.cpp
g++ -c print.cpp
g++ -c factorial.cpp
g++ -c multiply.cpp
g++ -o main main.o print.o factorial.o multiply.o
./main
```

1.2. In case of source code files located in different places, how can we write a Makefile?

Ta có thể sử dụng Recursive Make, có nghĩa là sử dụng `make` như một lệnh trong makefile. Kỹ thuật này rất hữu ích khi một dự án lớn chứa các thư mục con và mỗi thư mục chứa các makefile tương ứng.

Ví dụ:

```
/main
|_ Makefile
|_ /foo
    |_ Makefile
    |_ ... // other files
|_ /bar
    |_ Makefile
    |_ ... // other files
|_ /koo
    |_ Makefile
    |_ ... // other files
```

Để chạy các makefile của thư mục con từ bên trong makefile của main. Makefile của main sẽ có vòng lặp như dưới đây:

```
SUBDIRS = foo bar koo
```

```
subdirs:
    for dir in $(SUBDIRS); do \
        $(MAKE) -C $$dir; \
    done
```

Và khai báo trong các thư mục con dưới dạng .PHONY target:

```
SUBDIRS = foo bar koo
```

```
.PHONY: subdirs $(SUBDIRS)
```

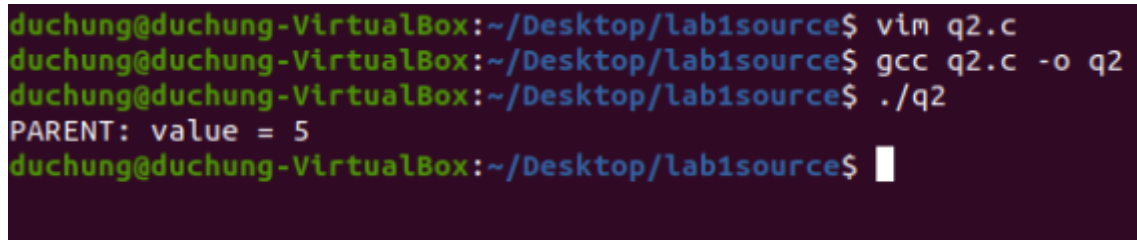
```
subdirs: $(SUBDIRS)

$(SUBDIRS):
    $(MAKE) -C $@
```

1.3. What the output will be at LINE A? Explain your answer.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5 ;
int main()
{
    pid_t pid ;
    pid = fork() ;
    if(pid == 0) { /* child process */
        value += 15;
        return 0 ;
    }
    else if (pid > 0) { /* parent process */
        wait (NULL) ;
        printf ( "PARENT:  value =%d" , value); /* LINE A */
        return 0 ;
    }
    return 0;
}
```

Output ở **LINE A** sẽ là **PARENT: value = 5** bởi vì, khi gặp lệnh `pid = fork()`, chương trình sẽ tạo ra 2 process cha và con, theo quy ước thì `pid == 0` là process con, `pid > 0` là process cha. Hai process cha và con hoạt động độc lập với nhau, như vậy process cha vẫn giữ được giá trị `int value = 5` khi được khai báo ở biến toàn cục.



```
duchung@duchung-VirtualBox:~/Desktop/lab1source$ vim q2.c
duchung@duchung-VirtualBox:~/Desktop/lab1source$ gcc q2.c -o q2
duchung@duchung-VirtualBox:~/Desktop/lab1source$ ./q2
PARENT: value = 5
duchung@duchung-VirtualBox:~/Desktop/lab1source$
```

2. BASIC COMMANDS

- 1 Create a new directory entitled with your student ID.
- 2 Create a new file `example.txt` under folder `<StudentID>` with the following content:
//Your student ID
To start vim editor, run the command: `vim`
The editor is now in command mode. To start editing the file content, enter: `:i[enter]`
To save: `:w`
To save and exit: `:wq`
To exit: `:q`
- 3 List the content of `<student-ID>` directory
- 4 View the content of file `example.txt`.
- 5 Show first 5 lines from `example.txt`.
- 6 Show last 5 lines from `example.txt`.
- 7 List all commands that have been run from that terminal session and save to `<StudentID>-history.txt`.
Students save "`<StudentID>-history.txt`" and "`example.txt`" in a folder.

```

duchung@duchung-VirtualBox:~/Desktop$ mkdir 2013381
duchung@duchung-VirtualBox:~/Desktop$ cd 2013381
duchung@duchung-VirtualBox:~/Desktop/2013381$ vim example.txt
duchung@duchung-VirtualBox:~/Desktop/2013381$ ls
example.txt
duchung@duchung-VirtualBox:~/Desktop/2013381$ cat example.txt
2013381
duchung@duchung-VirtualBox:~/Desktop/2013381$ head -5 example.txt
2013381
duchung@duchung-VirtualBox:~/Desktop/2013381$ tail -5 example.txt
2013381

```

3. PROGRAMMING EXERCISES

3.1. Problem 1

Write factorial.c to implement function factorial(): the function get an integer and return its factorial. For example factorial(6) returns 720.

```

// factorial.c
#include "factorial.h"
int factorial(const int aNumber){
    if(aNumber == 1)
        return 1;

    return aNumber * factorial(aNumber - 1);
}

```

3.2. Problem 2

Write readline.c to implement read_line(): read_line() gets data from stdin (keyboard), line-by-line. The content from stdin will be recorded on the parameter of this function named str. The result of read_line() indicates that whether the line is an integer or not. For example, with the input string below:

```

Hello , world
Operating system
Computer Science and Engineering
123

```

After calling the function, read_line() writes "Hello,world" into str and returns 0. The second calling will write "Operating system" into str and return 0. The third calling will write "Computer Science and Engineering" into str and return 0. The last call will write "123" into str and return 1.

```

// readline.c
#include "readline.h"

int read_line(char* str){
    int f = 1;
    int size = 0;
    while(str[size] != '\0') size++;

    for(int i = 0 ; i < size ; i++)
    {
        if(str[i] < 48 || str[i] > 57)
            f = 0;
    }

    return f;
}

```

3.3. Problem 3

Write `main.c` to create an executable file named **myfactorial** that reads input from stdin line by line and compute factorial if the line is an integer (each line does not exceed 50 letters). Then print factorial if the line is an integer else print -1. Write a Makefile to compile the program at least two targets:

- `all`: create `myfactorial` from other files.
- `clean`: remove all of object files, binaries

```
// main.c
#include <stdio.h>
#include "factorial.h"
#include "readline.h"

int main(){
    char str[1024];
    scanf("%s",str);
    if(read_line(str))
    {
        int num = 0,
            size = 0;

        while(str[size] != '\0') size++;

        for(int i = 0 ; i < size ; i++)
        {
            num *= 10;
            num += str[i] - '0';
        }
        printf("%d\n", factorial(num));
        return 0;
    }
    printf("%d\n", -1);
    return 0;
}
```

```
# makefile
all: main.o factorial.o readline.o
    gcc factorial.o main.o readline.o -o myfactorial
factorial.o: factorial.c factorial.h
    gcc -c factorial.c
main.o: main.c factorial.h readline.h
    gcc -c main.c
readline.o: readline.c readline.h
    gcc -c readline.c
clean:
    rm -f *.o myfactorial
```

```
duchung@duchung-VirtualBox:~/Desktop/source$ make all
gcc -c main.c
gcc -c factorial.c
gcc -c readline.c
gcc factorial.o main.o readline.o -o myfactorial
duchung@duchung-VirtualBox:~/Desktop/source$ ./myfactorial
6
720
duchung@duchung-VirtualBox:~/Desktop/source$ ./myfactorial
abc
-1
duchung@duchung-VirtualBox:~/Desktop/source$ make clean
rm -f *.o myfactorial
duchung@duchung-VirtualBox:~/Desktop/source$
```

3.4. Problem 4

Given a file named "numbers.txt" containing multiple lines of text. Each line is a non-negative integer. Write a C program that reads integers listed in this file and stores them in an array (or linked list). The program then uses the fork() system call to create a child process. The parent process will count the numbers of integers in the array that are divisible by 2. The child process will count numbers divisible by 3. Both processes then send their results to the stdout. For examples, if the file "numbers.txt" contains the following lines

```
12
3
4
10
11
```

```
// problem4.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

#define _MAX_SIZE 100

int main(){

    int  numArr[_MAX_SIZE];
    int  idx = 0;
    FILE *file;

    file = fopen("numbers.txt","r");

    while(fscanf(file, "%d", &numArr[idx]) != EOF)
    {
        idx++;
    }

    fclose(file);

    int  ArrSize = idx;
    int  count = 0;

    pid_t pid = fork();

    if(pid == 0)
    {
        // child process
        for(int i = 0 ; i < ArrSize ; i++)
        {
            if(numArr[i] % 3 == 0)
                count++;
        }
        printf("%d\n", count);
    }
    else
    {
        // parent process
        for(int i = 0 ; i < ArrSize ; i++)
        {
            if(numArr[i] % 2 == 0)
                count ++;
        }
    }
}
```

```
    }  
    printf("%d\n", count);  
    wait(NULL);  
}  
return 0;  
}
```

```
duchung@duchung-VirtualBox:~/Desktop/lab1source$ gcc ./problem4.c -o problem4  
duchung@duchung-VirtualBox:~/Desktop/lab1source$ ./problem4  
3  
2  
duchung@duchung-VirtualBox:~/Desktop/lab1source$
```

Tài liệu

- [1] MakeFile in C++ and its applications
<https://goeco.link/rHGfq>
- [2] Using .PHONY for recursive invocations of 'make' command
<https://goeco.link/SmBeG>
- [3] Recursive Use of make
https://www.tack.ch/gnu/make-3.82/make_43.html