Experiment3 C programming experiment

Experimental purpose:

Further use the basic syntax of C programming language in Linux system, deepen the understanding of the
knowledge.

(1) Task 1

(1) Write a C program that uses standard I/O libraries to display the contents of text files. The program is
compiled and linked by the make tool, which requires the generation of the.o file first, and then the generation
of the executable file, and the function of deleting the intermediate file (.o) in the makefile file.

c1:c:

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    char buf[1024] = { 0 };
    FILE* fp = fopen(argv[1],"r");
    if (argc < 2)
    {
        printf("please input source file!\n");
    }
    if (fp == NULL)
    {
        printf("open source %s failed\n", argv[1]);
        return -1;
    }
    while (fgets(buf,1024, fp))
    {
        printf("%s\n", buf);
    }
    return 0;
}
```

This C code reads a text file line by line and prints each line to the console. The filename is provided as a command-line argument. If the file cannot be opened or no filename is provided, an error message is displayed.

Makefile:

```
hello1:c1.o
	gcc -o hello1 c1.o
c1.o:c1.c
	gcc -c c1.c
clean:
	rm -rf *.o
```

This Makefile is used for automating the process of compiling and linking a C program.

Targets:

hello1:c1.o: This is the target to build the executable hello1. It depends on the object file c1.o. When make hello1 is run, it will check if c1.o is up to date and, if not, will compile it and then link it to create hello1.
c1.o:c1.c: This target defines how to compile c1.c into an object file c1.o. It tells make to run gcc -c c1.c, which compiles the c1.c source code into an object file.

Commands:

gcc -o hello1 c1.o: This command links c1.o into the final executable hello1.
gcc -c c1.c: This compiles c1.c into c1.o (an object file) but does not link it yet.

Clean:

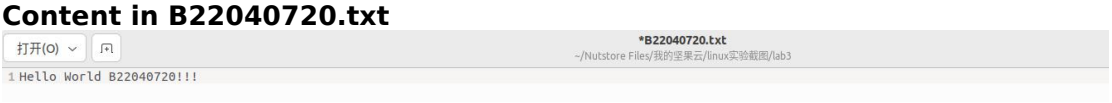
clean: This target is used to remove all object files (*.o) and any other generated files. When you run make clean, it will remove the object files and allow for a fresh build.

Output:

```
yyy@canghaihuaovo:~/文档/实验/linux实验三$ touch Makefile
yyy@canghaihuaovo:~/文档/实验/linux实验三$ make
gcc -c c1.c
gcc -o hello1 c1.o
yyy@canghaihuaovo:~/文档/实验/linux实验三$ ./hello1 B22040720yinyuyang.txt
Hello WOrld B22040720!!!

yyy@canghaihuaovo:~/文档/实验/linux实验三$ make clean
rm -rf *.o
yyy@canghaihuaovo:~/文档/实验/linux实验三$
```

Content in B22040720.txt



(2) Task 2

(1) Write a C program that displays all the file names in the current directory. The program is compiled and
linked by the make tool, which requires the generation of the.o file first, and then the generation of the
executable file, and the function of deleting the intermediate file (.o) in the makefile file.

c2.c:

```c
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>
int main(int argc, char* argv[])
{
DIR* dirp;
struct dirent* direntp;
if ((dirp = opendir(argv[1])) == NULL) {
printf("error\n");
//
exit(1);
}
while ((direntp = readdir(dirp)) != NULL)
printf("%s\n", direntp->d_name);
closedir(dirp);
// exit(0);

}
```

This C code opens a directory specified as a command-line argument. It then reads the directory's contents, printing the name of each file or subdirectory. If there's an error opening the directory, an error message is printed and the program exits.

Makefile:

```
hello2: c2.o
	gcc -o hello2 c2.o

c2.o: c2.c
	gcc -c c2.c

clean:
	rm -rf *.o hello2
```

Targets:

hello2:c2.o: This target builds the executable hello2, and it depends on c2.o. When you run make hello2, it checks if c2.o is up to date. If not, it compiles c2.c and then links c2.o to create the hello2 executable.
c2.o:c2.c: This target specifies how to compile c2.c into c2.o. It uses the command gcc -c c2.c to generate the object file.

Commands:

gcc -o hello2 c2.o: This command links c2.o to create the hello2 executable.
gcc -c c2.c: This compiles c2.c into the object file c2.o.

Clean:

clean: This target removes all .o object files when you run make clean, helping you keep the project directory clean.

Output:

```
yyy@canghaihuaovo:~/文档/实验/linux实验三$ sudo vim Makefile
yyy@canghaihuaovo:~/文档/实验/linux实验三$ make
gcc -c c2.c
gcc -o hello2 c2.o
```

```
yyy@canghaihuaovo:~/文档/实验/linux实验三$ ./hello2 /home/yyy/文档/实验/linux实
验三
Makefile
.
..
c2.c
c2.o
hello2
.c2.c.swp
```

(3) Task 3

(3) Write a C program that changes the working directory of the current process. The program is compiled
and linked by the make tool, which requires the generation of the.o file first, and then the generation of the
executable file, and the function of deleting the intermediate file (.o) in the makefile file.

c3.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
char buf[1024] = {0};char buf2[1024]={0};
getcwd(buf, 1024);
printf("%s\n", buf);
if(chdir("/home")<0){
printf("error\n");
}
else
{
printf("success\n");
}
getcwd(buf2,1024);
printf("%s\n",buf2);
return 0;
}
```

It uses the getcwd function to get the current directory where the program is running and stores it in the buf variable.It then prints the obtained directory to the console using printf.
It tries to change the current directory to /home using the chdir function.

If the directory change is unsuccessful, it prints an error message.

If the directory change is successful, it gets the new current directory using getcwd again and stores it in buf2.

Makefile:

```
hello3:c3.o
	gcc -o hello1 c3.o
c3.o:c3.c
	gcc -c c3.c
clean:
	rm -rf *.o
```

Targets:

hello3:c3.o: This target builds the hello3 executable from the c3.o object file. When you run make hello3, it checks if c3.o is up to date. If not, it will compile c3.c and then link c3.o to create hello3.

c3.o:c3.c: This target compiles c3.c into c3.o using gcc -c c3.c.

Commands:

gcc -o hello3 c3.o: Links c3.o to create the hello3 executable.

gcc -c c3.c: Compiles c3.c into the object file c3.o.

Clean:

clean: Removes all .o object files when you run make clean.

Output: