

CS 314 Assignment - 2

Kavali Sri Vyshnavi Devi (200010023)

January 15, 2023

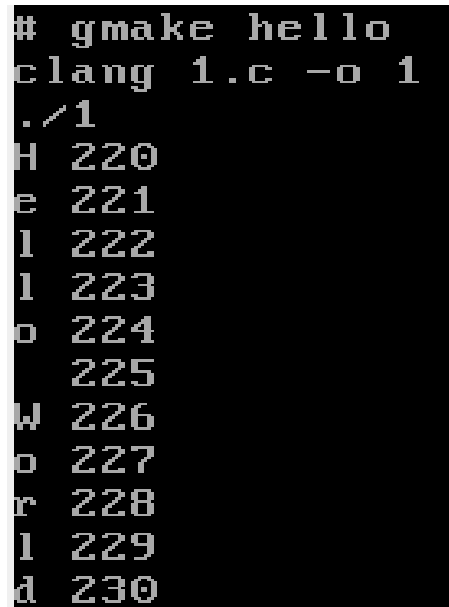
1 Part - 1

The minimum lines of C code required are 23. This number also includes the lines of code written to include different libraries. Without including the lines of library there are a total of 18 lines.

The code is tested on minix3 operating system using the compiler named clang.

command to execute the code is `gmake hello`.

The following shows the result obtained by execution of the code using the above mentioned command.



```
# gmake hello
clang 1.c -o 1
./1
H 220
e 221
l 222
l 223
o 224
  225
W 226
o 227
r 228
l 229
d 230
```

Figure 1: Executing the program

2 Part - 2

In this part 3 C programs are written such namely half.c, twice.c and square.c. Along with these files a makefile is also present which makes executable of all c program files present in the directory with names hald, twice and square.

Each of the C programs use **execvp** command to execute according to the arguments given as mentioned in the example \$./twice ./square ./half 100

int execvp (const char *file, char *const argv[]);

execvp command takes file to execute and all the args to the file as arguments.

Here the technique used to execute according to the above mentioned example is the first argument file will be executing the operation on last argument that is the number provided and all the remaining are the arguments, last argument is the one on which operations should be performed. So at last of each program execvp is called until the argument left is only one, for each execvp call all the arguments are passed except the first one as that got executed in the present program itself. Also after operating on the last argument that is the integer given to operate it is converted from string to integer, performs operation according to the file on the integer later converts it into string so that again it is sent in the arguments list to next file to perform next operation.

Command to execute the code is gmake.

This command compiles all the program files present in the folder and later those can be used to executed as like in example provided.

```
# ls
half.c  makefile  square.c  twice.c
# gmake
clang square.c -o square
clang half.c -o half
clang twice.c -o twice
# ls
half    half.c  makefile  square  square.c  twice  twice.c
# ./twice ./square ./half ./twice ./half 10
Twice: Current process id: 7994, Current result: 20
Square: Current process id: 7994, Current result: 400
Half: Current process id: 7994, Current result: 200
Twice: Current process id: 7994, Current result: 400
Half: Current process id: 7994, Current result: 200
```

Figure 2: Execution

3 Part - 3

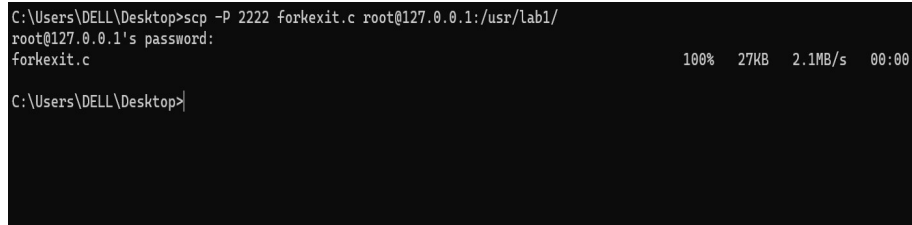
A process is basically a program in execution. The set of processes forms a tree via the fork system call. New processes are created by the fork system call. With the exception of the value returned, the new process that fork() creates is a duplicate of the existing one. The parent process creates child process and scheduler schedules these processes accordingly. If wait system call is used then parent waits till the child process gets terminated in that case child exits first and they will get exited in the reverse order of which they are created. This order is justified because the parent process must wait for the child process to exit before terminating itself. Otherwise, it will not be able to spawn a child process.

The forkexit.c file located in minix/servers/pm has been changed, changes are made in *do_fork()* function and in *do_exit()* function.

The below lines of code is added in forkexit.c file in *do_fork()* and *do_exit()* functions respectively.

```
printf("Minix: PID %d created\n", rmc->mp_pid);  
printf("Minix: PID %d exited\n", mp->mp_pid);
```

The file was sent from guest operating system (Windows) to host operating system (minix) through SCP command.



```
C:\Users\DELL\Desktop>scp -P 2222 forkexit.c root@127.0.0.1:/usr/lab1/  
root@127.0.0.1's password:  
forkexit.c 100% 27KB 2.1MB/s 00:00  
C:\Users\DELL\Desktop>
```

Figure 3: SCP from host OS to guest OS

On successful build after making changes in forkexit.c and rebooting the changes will be reflected in the minix operating system.

The below figures show the building of OS and it's successful completion.



```
# bash run.sh  
copying files  
going to src directory and building the update code
```

Figure 4: Executing run.sh file

```
dependall ==> usr.bin/tsort
dependall ==> usr.bin/tty
dependall ==> usr.bin/ul
dependall ==> usr.bin/uname
dependall ==> usr.bin/unexpand
dependall ==> usr.bin/unifdef
dependall ==> usr.bin/uniq
dependall ==> usr.bin/units
dependall ==> usr.bin/unvis
dependall ==> usr.bin/unzip
dependall ==> usr.bin/users
dependall ==> usr.bin/uudecode
dependall ==> usr.bin/uuencode
dependall ==> usr.bin/uuidgen
dependall ==> usr.bin/vis
dependall ==> usr.bin/w
dependall ==> usr.bin/wall
dependall ==> usr.bin/wc
dependall ==> usr.bin/what
```

Figure 5: Executing run.sh file

```
Done.
Build started at: Sat Jan 14 05:59:11 IST 2023
Build finished at: Sat Jan 14 06:06:15 IST 2023
```

Figure 6: Completion of building the OS

```
# reboot
```

Figure 7: Rebooting the OS

After rebooting the operating system when a process is created then PID is printed (along with created message), also during exit of the program PID is printed (along with exited message).

```
# ls
Minix: PID 215 created
bin      boot_monitor home      mnt      sbin      tmp
boot     dev          lib       proc     service   usr
boot.cfg etc          libexec   root     sys       var
Minix: PID 215 exited
```

Figure 8: Executing ls command

```
# printenv
Minix: PID 1724 created
PWD=/usr/src
LD_LIBRARY_PATH=/usr/local/lib:/usr/pkg/lib:/usr/lib:/lib
HOME=/root
PATH=/usr/local/sbin:/usr/pkg/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/pkg/bin:/usr/bin:/bin:/usr/games
TERM=minix
OLDPWD=/usr/lab1/q3
USER=root
PAGER=less
TZ=Asia/Calcutta
EDITOR=vi
LOGNAME=root
SHELL=/bin/sh
Minix: PID 1724 exited
```

Figure 9: Executing printenv command