$ gcc -E hello.c

$ gcc -E hello.c 1> hello.i

To do pre processing

$ gcc -S hello.i

To do compilation

$gcc -S hello.i -std=c11

$ man 7 standards

To check different standards in C

$ gcc -c hello.s

To do assembly phase, converting the assembly code to machine code.

$ gcc -c -ggdb hello.s

To add the debugging symbols in the output file.

$ gcc hello.o

To do linking phase(dynamic by default). By default a.out executable file is created.

$ gcc hello.o -o dynamicexe.out -lc

To get the executable file with different names.

$ gcc hello.o -o staticexe.out -lc --static

$ ls -lh dynamicexe.out staticexe.out

To get the sizes of files. Static executables(it contains complete code) occupies more space compare to dynamic executables(it do not contains the code of external functions, ex: printf()).

$ man readelf

$ readelf -a hello.o

To view the contents of object files. <-h> displays head information. <-S> to view the names of different sections in object file.

$ man objdump

$ objdump -h hello.o

<-D> to get assembly of all the sections. <-d> to get assembly on main. By default it shows AT&T format.

$ objdump -d -M intel hello.o

To get assembly in intel format.

$ gcc -ggdb hello.c -o hello.out

$ gdb -q ./hello.out

To open gdb debugger in quite mode

(gdb) disassemble main

To get assembly of main function. By default assembly will be in AT&T format

(gdb) set disassembly-flavor intel

(gdb) disassemble main

Not flavor will set to intel format.

(gdb) break main

To add brakes on function level

(gdb) break 7

To add brakes on line level

(gdb) list

To see the program

(gdb) run

To run the program

(gdb) info registers

To get the numerical registers details

(gdb) info all-registers

To get all registers details

(gdb) c

To continue the code till end or next break pointer.

(gdb) n

To come to next line. Once after hitting n directly press enter key to go to next line.

(gdb) q

To exit

$ man 2 intro

To get information of system calls

$ man syscalls

To get list of system calls

/<write>

To search write system call in man page.

$ man 2 write

To get write system call man page.

**Refer ‘wraper.c’ program**

$ gcc wrapper.c -o wrapper.out

Compilation

$ ./wrapper.out

Execution

$ echo $?

To get the value return by the child program to its parent i.e, shell.

Note: we will use system calls by calling their appropriate wrapper functions. If we don’t want to use wrapper functions we can use ‘syscall’ system call.

$ man syscall

Note: we can pass six argument to any system call to the max using these registers. In case if we ant to pass more arguments to our system calls we can use pointers. Refer man pager for more details.

$ less /usr/include/x86\_64-linux-gnu/asm/unistd\_64.h

$ less /usr/include/asm/unistd\_64.h

Location where we can find system call numbers.

Note: if we want to use system call without using wrappers then we need to use system call numbers.

**Refer ‘syscall.c’ program**

$ gcc syscall.c -o syscall.out

$ ./syscall.out

$ echo $?

Invoking system call within an assembly code to actually understand how the arguments are placed in the registers.

**Refer ‘syscallFromAssy.nasm’ program**

To assemble the assembly code.

$ nasm -f elf64 syscallFromAssy.nams

This till generate syscallFromAssy.o file

$ ld syscallFromAssy.o -o syscallFromAssyLD.out

(or)

$ gcc syscallFromAssy.o -o syscallFromAssyGCC.out

$ ./ syscallFromAssyLD.out

$ echo $?

$ ./ syscallFromAssyGCC.out

$ echo $?













