Module: R1: C Programming
Section: Build Systems **Task:** Source to Binary

Task
Source to Binary

The **`main.c`** I will be using throughout this task is given by the following program:

```
#include <stdio.h>

int main() {

printf("Hello World!\n");

printf("My first makefile!\n");

return 0;

}
```

Makefile:

I will just explain the contents of the Makefile first and then proceed to the C compilation pipeline.

1. **CC = gcc:** It defines the C compiler to be usedas gcc.
2. **all: preprocessor compiler assembler linker run:**It specifies the target all, which depends on preprocessor, compiler, assembler, linker, and run. When we run make all, it will execute all these steps in sequence.
3. **PHONY: all clean:** It declares all and clean as phony targets. Phony targets are those that are not actual files but are simply names for tasks. This is basically used to avoid any confusions.
4. **build: preprocessor compiler assembler linker:** It specifies the build target, which depends on preprocessor, compiler, assembler, and linker. As this target is not used in the all target, so we have to explicitly invoke ***make build*** to run these steps.
5. **run:** It executes the compiled program ***a.out***.
6. **linker:** It compiles **main.c** and links it to produce an executable named a.out.
7. **assembler:** It assembles **main.s** to produce an object file **main.o**.
8. **compiler:** It compiles **main.c** to produce assembly code **main.s**.

9. **preprocessor:** It runs the C preprocessor on **main.c** to produce preprocessed output.
10. **clean:** Removes all generated files (.exe, .o, .s, .out) and clears the terminal.

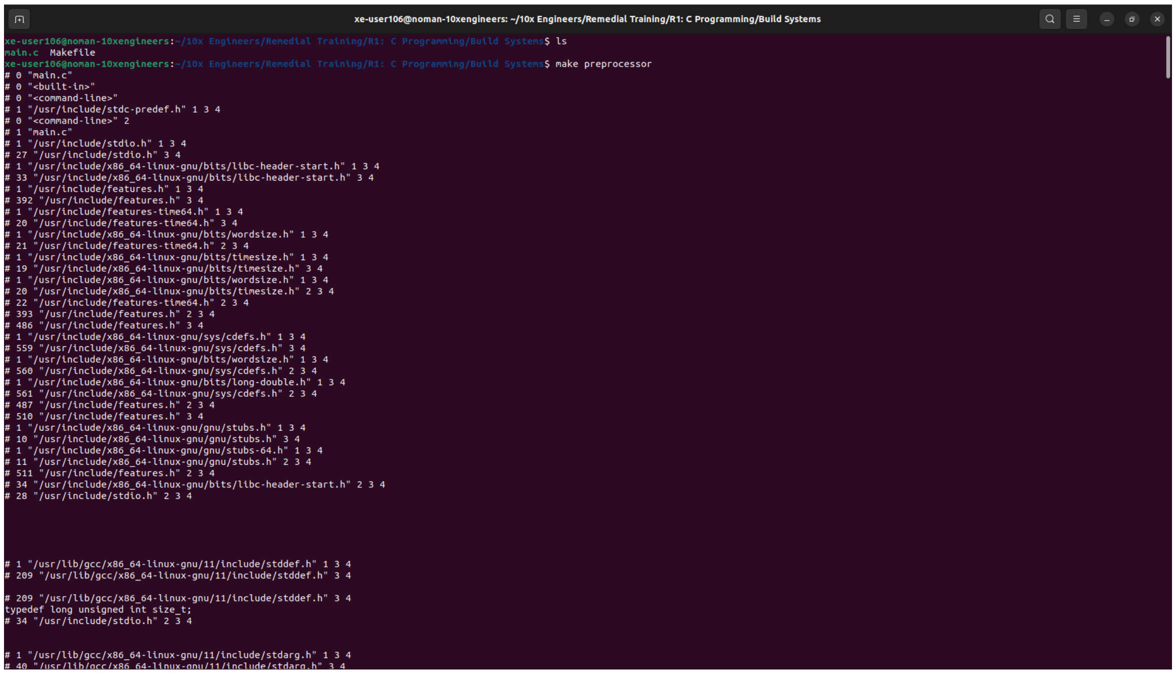
Compilation Pipeline:

1. **Preprocessing:**

First, let's preprocess main.c using the C preprocessor:

```
make preprocessor
```

This command will generate a preprocessed version of main.c. Let's take a look at the contents of the preprocessed file. Here's the output:



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make preprocessor
# 0 "main.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "main.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 392 "/usr/include/features.h" 3 4
# 1 "/usr/include/features-time64.h" 1 3 4
# 20 "/usr/include/features-time64.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 21 "/usr/include/features-time64.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 1 3 4
# 19 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 20 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 2 3 4
# 22 "/usr/include/features-time64.h" 2 3 4
# 393 "/usr/include/features.h" 2 3 4
# 486 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 559 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 560 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
# 561 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 487 "/usr/include/features.h" 2 3 4
# 510 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
# 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
# 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
# 511 "/usr/include/features.h" 2 3 4
# 34 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 2 3 4
# 28 "/usr/include/stdio.h" 2 3 4





# 1 "/usr/lib/gcc/x86_64-linux-gnu/11/include/stddef.h" 1 3 4
# 209 "/usr/lib/gcc/x86_64-linux-gnu/11/include/stddef.h" 3 4

# 209 "/usr/lib/gcc/x86_64-linux-gnu/11/include/stddef.h" 3 4
typedef long unsigned int size_t;
# 34 "/usr/include/stdio.h" 2 3 4


# 1 "/usr/lib/gcc/x86_64-linux-gnu/11/include/stdarg.h" 1 3 4
# 40 "/usr/lib/gcc/x86_64-linux-gnu/11/include/stdarg.h" 3 4
```

As we can see, it displays the preprocessed contents of **main.c**, where all preprocessor directives (such as **#include** and **#define**) have been processed.

In this file, we can also see the contents of the **<stdio.h>** header file included and any other preprocessor directives processed.

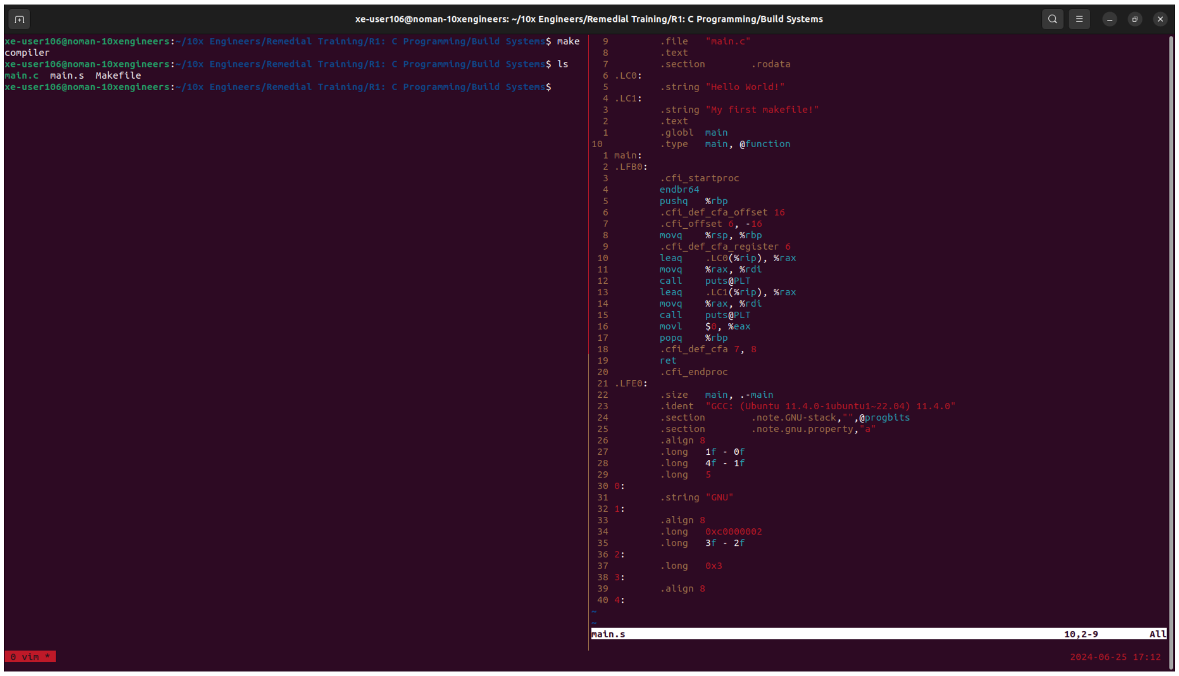
2. **Compiler:**

Let's compile the preprocessed **main.c** file using the C compiler specified in the Makefile.

`make compiler`

This command invokes the C compiler to generate assembly code from the preprocessed **main.c**. The output of this command produced an assembly file named **main.s**, which contains the assembly code equivalent to the C code in **main.c**.

Let's examine the contents of the generated assembly file. Here's the output:



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make
compiler
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$
```

```
        .file   "main.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello World!"
.LC1:
        .string "My first makefile!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        leaq    .LC1(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

3. **Assembler:**

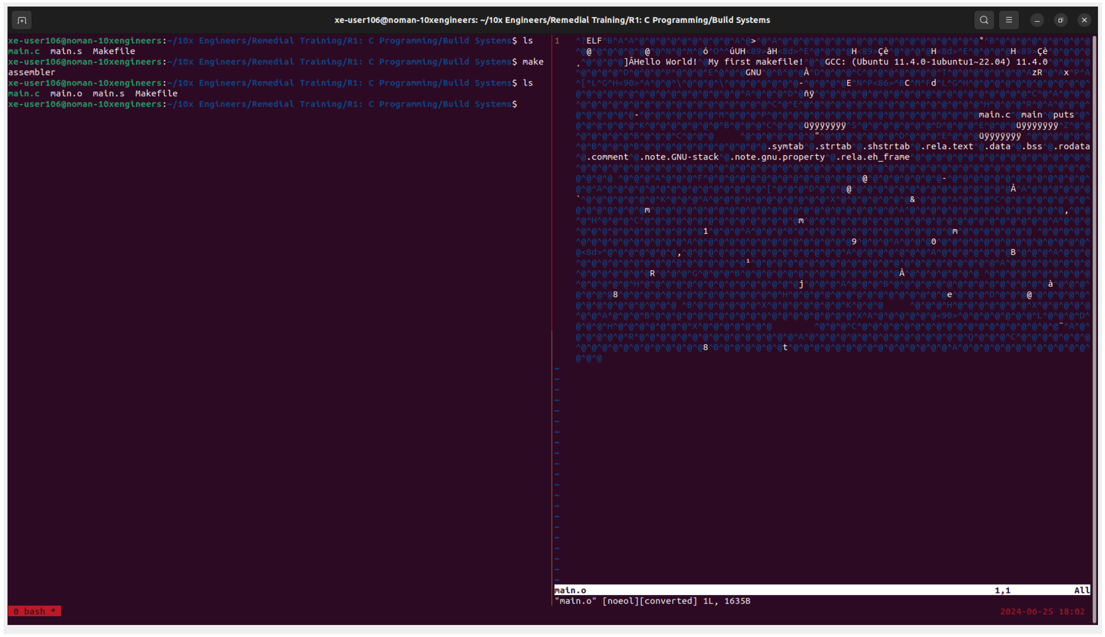
The assembling step involves translating the assembly code into machine code (object file).

Let's assemble the generated **main.s** file using the assembler specified in the Makefile.

`make assembler`

This command invokes the assembler to generate an object file from the assembly file **main.s**.

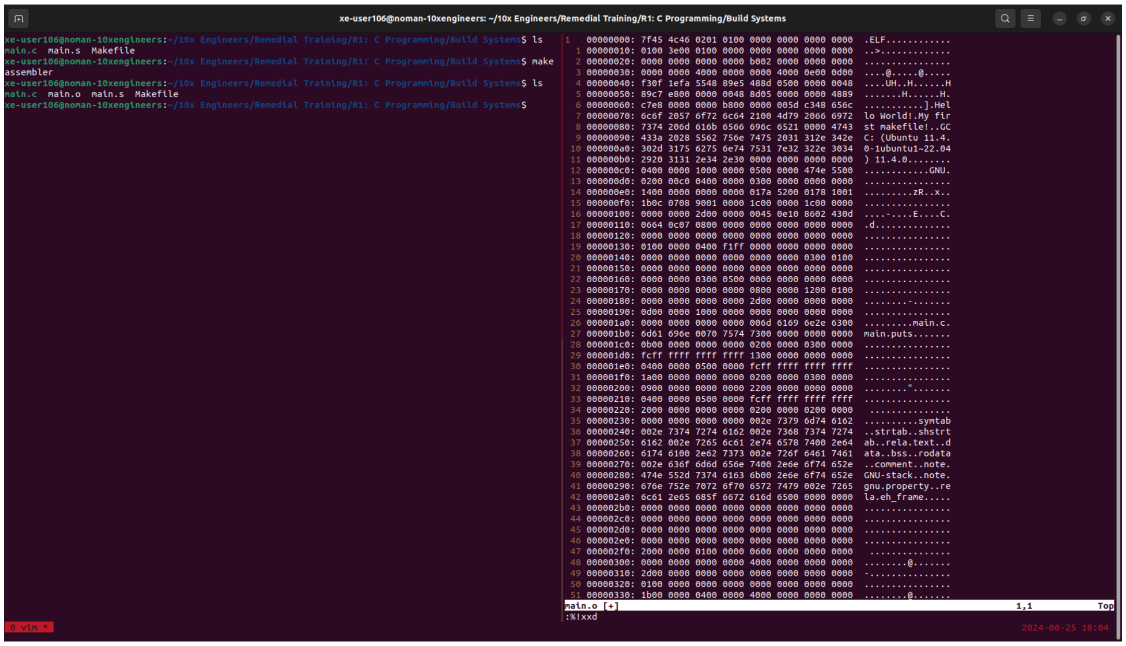
Let's examine the contents of the generated object file. Here's the output:



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make
assembler
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  main.o  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$
```

```
^?ELF ... Hello World! My first makefile! GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0 ... GNU ...
... main.c main puts ...
... .symtab .strtab .shstrtab .rela.text .data .bss .rodata
.comment .note.GNU-stack .note.gnu.property .rela.eh_frame ...
main.o                                                        1,1           All
"main.o" [noeol][converted] 1L, 1635B
```

We can convert this binary to hexadecimal information using ***%!xxd*** command. Here's a demonstration:



```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000  ..>.............
00000020: 0000 0000 0000 0000 b002 0000 0000 0000  ................
00000030: 0000 0000 4000 0000 0000 4000 0e00 0d00  ....@.....@.....
00000040: f30f 1efa 5548 89e5 488d 0500 0000 0048  ....UH..H......H
00000050: 89c7 e800 0000 0048 8d05 0000 0000 4889  .......H......H.
00000060: c7e8 0000 0000 b800 0000 005d c348 656c  ...........].Hel
00000070: 6c6f 2057 6f72 6c64 2100 4d79 2066 6972  lo World!.My fir
00000080: 7374 206d 616b 6566 696c 6521 0000 4743  st makefile!..GC
00000090: 433a 2028 5562 756e 7475 2031 312e 342e  C: (Ubuntu 11.4.
000000a0: 302d 3175 6275 6e74 7531 7e32 322e 3034  0-1ubuntu1~22.04
000000b0: 2920 3131 2e34 2e30 0000 0000 0000 0000  ) 11.4.0........
000000c0: 0400 0000 1000 0000 0500 0000 474e 5500  ............GNU.
000000d0: 0200 00c0 0400 0000 0300 0000 0000 0000  ................
000000e0: 1400 0000 0000 0000 017a 5200 0178 1001  .........zR..x..
000000f0: 1b0c 0708 9001 0000 1c00 0000 1c00 0000  ................
00000100: 0000 0000 2d00 0000 0045 0e10 8602 430d  ....-....E....C.
00000110: 0664 0c07 0800 0000 0000 0000 0000 0000  .d..............
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0300 0500 0000 0000 0000 0000  ................
00000170: 0000 0000 0000 0000 0800 0000 1200 0100  ................
00000180: 0000 0000 0000 0000 2d00 0000 0000 0000  ........-.......
00000190: 0d00 0000 1000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 006d 6169 6e2e 6300  .........main.c.
000001b0: 6d61 696e 0070 7574 7300 0000 0000 0000  main.puts.......
000001c0: 0b00 0000 0000 0000 0200 0000 0300 0000  ................
000001d0: fcff ffff ffff ffff 1300 0000 0000 0000  ................
000001e0: 0400 0000 0500 0000 fcff ffff ffff ffff  ................
000001f0: 1a00 0000 0000 0000 0200 0000 0300 0000  ................
00000200: 0900 0000 0000 0000 2200 0000 0000 0000  ........".......
00000210: 0400 0000 0500 0000 fcff ffff ffff ffff  ................
00000220: 2000 0000 0000 0000 0200 0000 0200 0000   ...............
00000230: 0000 0000 0000 0000 002e 7379 6d74 6162  ..........symtab
00000240: 002e 7374 7274 6162 002e 7368 7374 7274  ..strtab..shstrt
00000250: 6162 002e 7265 6c61 2e74 6578 7400 2e64  ab..rela.text..d
00000260: 6174 6100 2e62 7373 002e 726f 6461 7461  ata..bss..rodata
00000270: 002e 636f 6d6d 656e 7400 2e6e 6f74 652e  ..comment..note.
00000280: 474e 552d 7374 6163 6b00 2e6e 6f74 652e  GNU-stack..note.
00000290: 676e 752e 7072 6f70 6572 7479 002e 7265  gnu.property..re
000002a0: 6c61 2e65 685f 6672 616d 6500 0000 0000  la.eh_frame.....
000002b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002f0: 2000 0000 0100 0000 0600 0000 0000 0000   ...............
00000300: 0000 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000310: 2d00 0000 0000 0000 0000 0000 0000 0000  -...............
00000320: 0100 0000 0000 0000 0000 0000 0000 0000  ................
00000330: 1b00 0000 0400 0000 4000 0000 0000 0000  ........@.......
main.o [+]                                                    1,1           Top
:%!xxd
```

The output of the **make assembler** command produced an object file named **main.o**, which contains the machine code generated from the assembly code in **main.s**.

4. **Linker:**

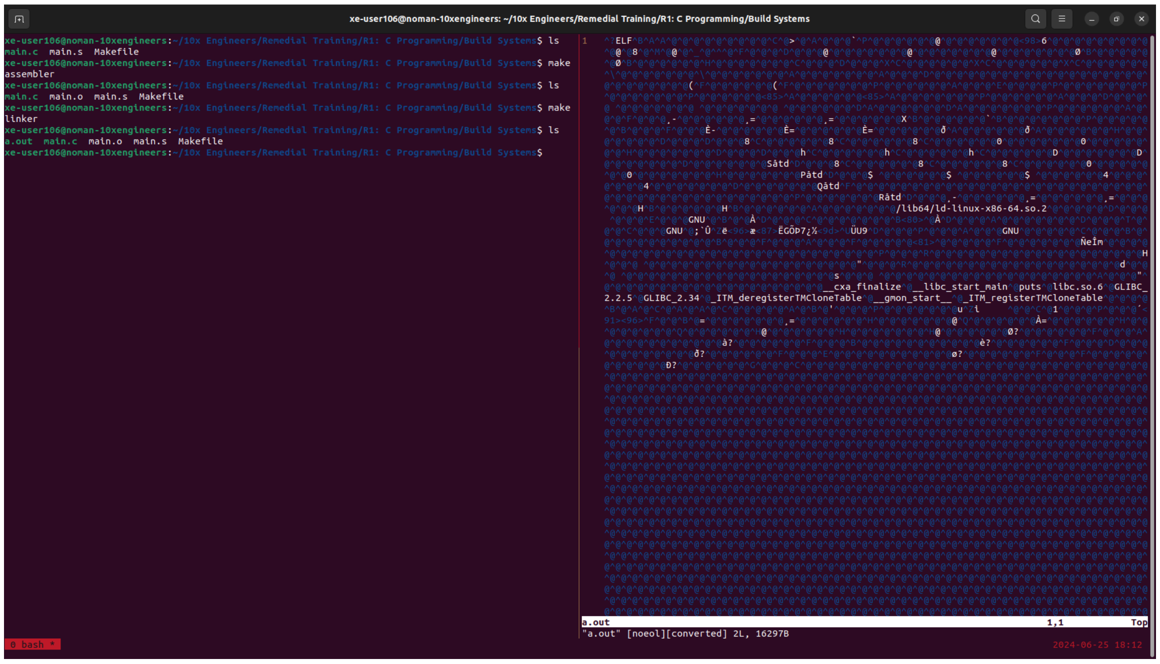
The linking step involves combining the object file generated from our source file with any necessary system libraries to create an executable program.

In our case, we'll link the object file **main.o** to produce the final executable program.

`make linker`

This command invokes the linker to combine the object file **main.o** with necessary system libraries to generate the executable program. Let's examine the contents of the generated executable file.

Here's the output:



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make
assembler
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
main.c  main.o  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make
linker
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
a.out  main.c  main.o  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$
```

```
a.out                                                    1,1           Top
"a.out" [noeol][converted] 2L, 16297B
```

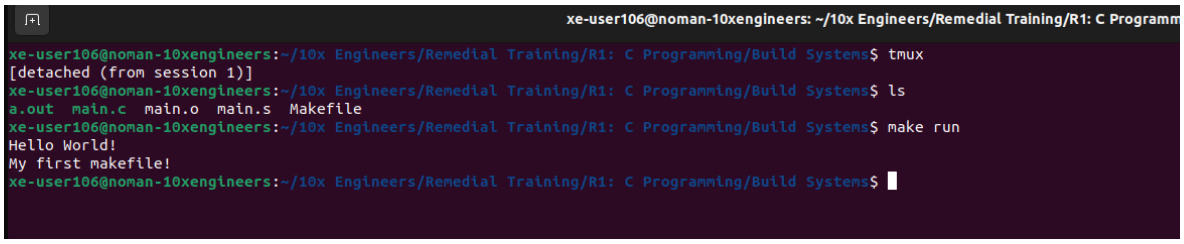
The output of this command produced an executable file named **a.out**, which contains the linked program ready for execution.

5. **Run:**

Now, let's proceed to run the generated executable to verify that it behaves as expected.

`make run`

This command will execute the **a.out** file, which is our compiled program. Here's the output:



```
xe-user106@noman-10xengineers: ~/10x Engineers/Remedial Training/R1: C Programm
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ tmux
[detached (from session 1)]
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ ls
a.out  main.c  main.o  main.s  Makefile
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$ make run
Hello World!
My first makefile!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Build Systems$
```

This output confirms that our program executed successfully.