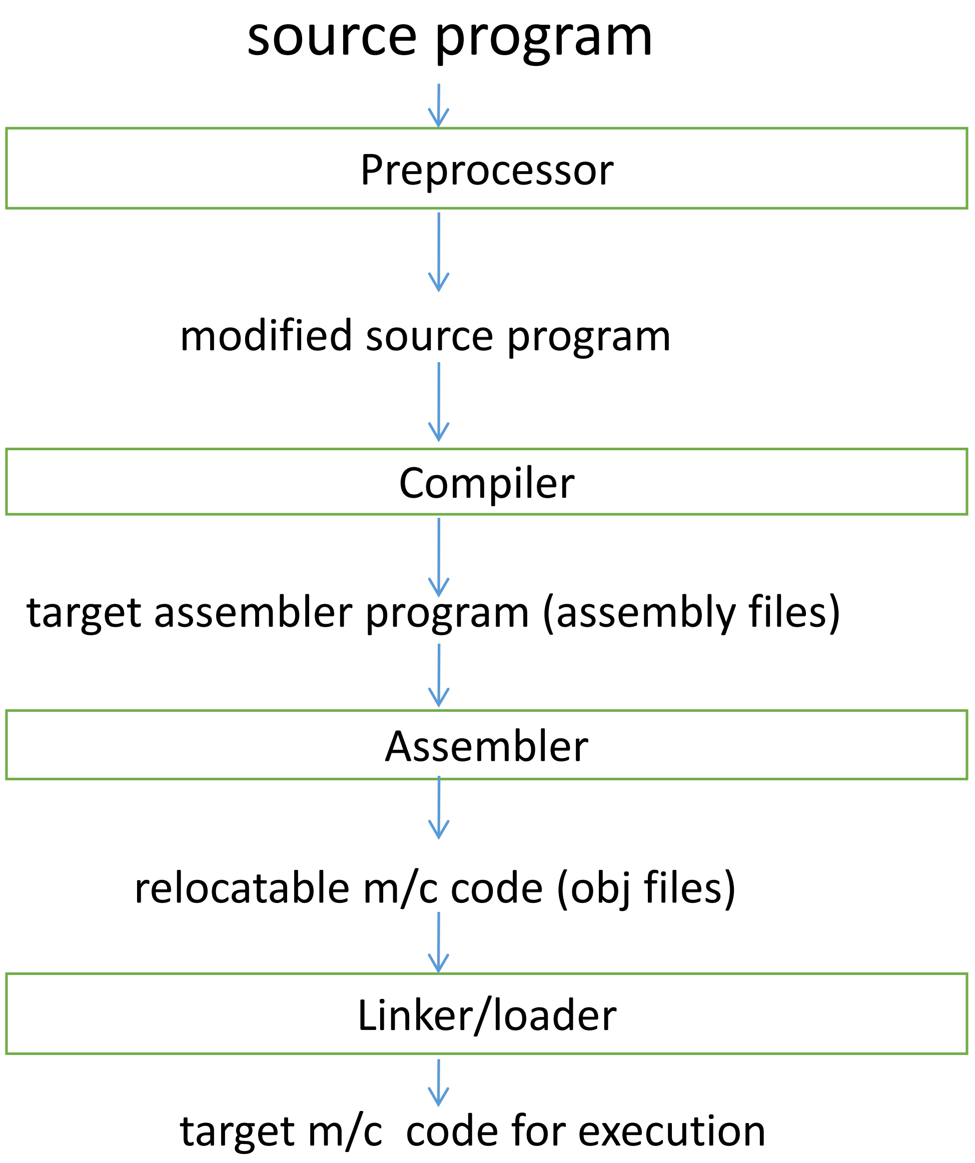
Linking and Loading

Write a program, use gcc to compile, and you will get an executable

It is pretty simple. Right?

What happens during the compilation process and how the program gets converted to an executable?



source program

Preprocessor

modified source program

Compiler

target assembler program (assembly files)

Assembler

relocatable m/c code (obj files)

Linker/loader

target m/c code for execution

What goes inside the compilation process?

- Compiler converts a program to an executable. There are four phases for a C program to become an executable:
  - Pre-processing
  - Compilation
  - Assembly
  - Linking

Example

- Consider a C program having main() and add()

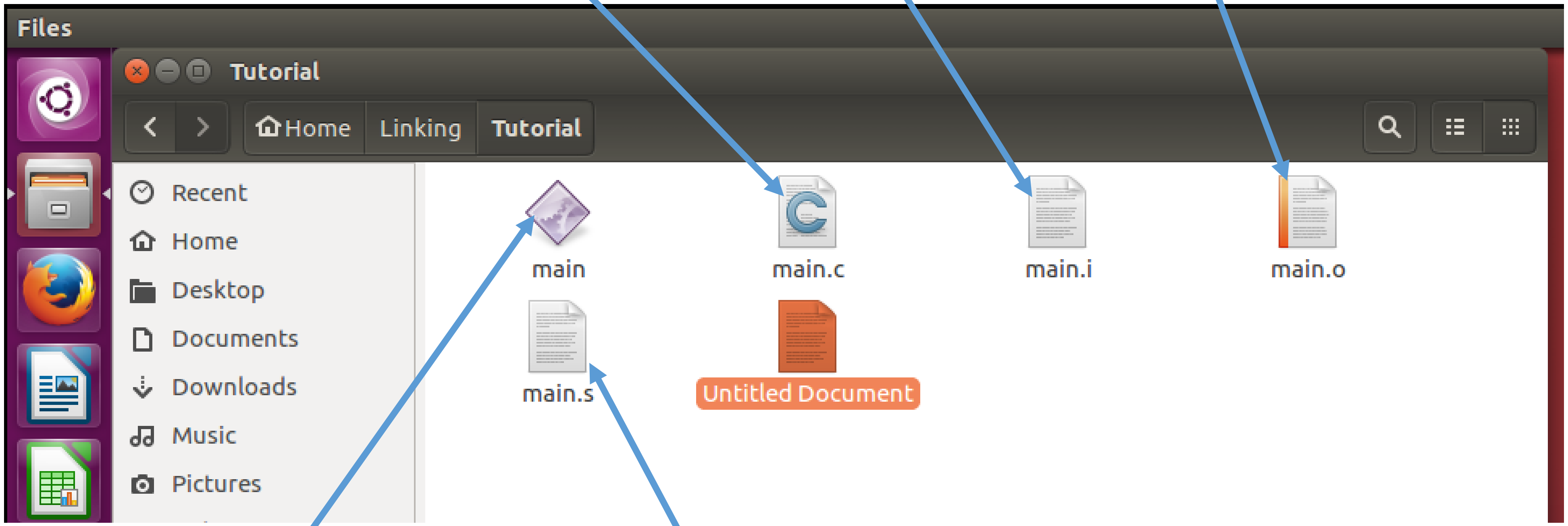
```
#include<stdio.h>

    int add();

    int main(){
       add();
    }

    int add(){
       int num1=5, num2=6, sum;
       sum=num1+num2;
      printf("SUM=%d",sum);
      return 0;
    }
```

Different intermediate files



source file main.c

intermediate file main.i

object file main.o

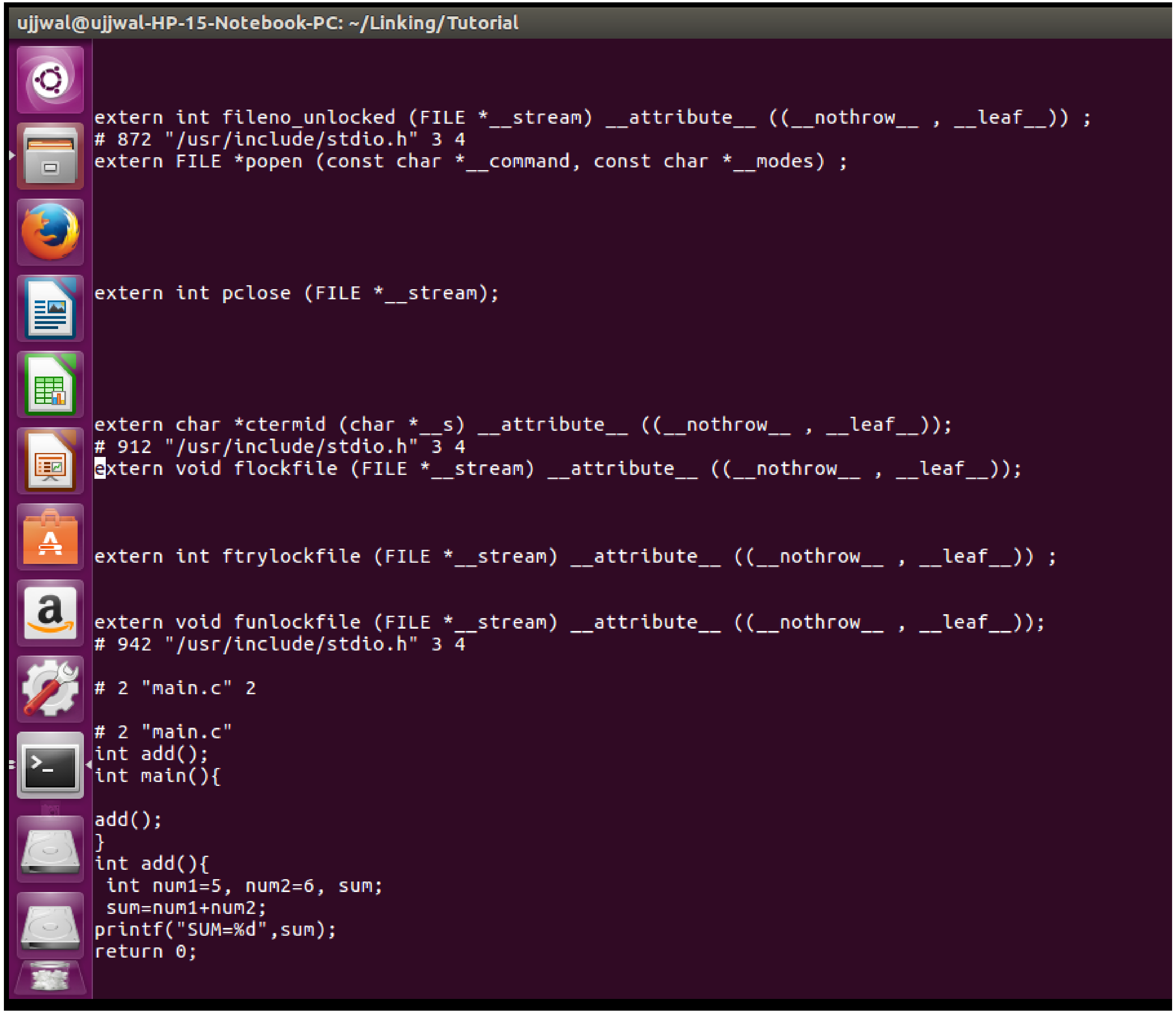
executable file main

assembly file main.s

Pre-processing

- This is the first phase of compilation process. This phase include:
    - Removal of Comments
    - Expansion of Macros
    - Expansion of the included files
- The preprocessed output is stored in the **filename.i**.
    - Translates the C source file main.c into an intermediate file main.i

Let’s see what’s inside filename.i: using **$vi filename.i**



```
extern int fileno_unlocked (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;
# 872 "/usr/include/stdio.h" 3 4
extern FILE *popen (const char *__command, const char *__modes) ;





extern int pclose (FILE *__stream);





extern char *ctermid (char *__s) __attribute__ ((__nothrow__ , __leaf__));
# 912 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));



extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;


extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));
# 942 "/usr/include/stdio.h" 3 4

# 2 "main.c" 2

# 2 "main.c"
int add();
int main(){

add();
}
int add(){
 int num1=5, num2=6, sum;
 sum=num1+num2;
printf("SUM=%d",sum);
return 0;
```

Compiling

- The next step is to compile filename.i and produce an intermediate file **filename.s**. This file is in assembly level instructions.
  - Translates main.i into an assembly language file main.s

- Let’s see through this
- file using **$vi filename.s**
- The snapshot shows that it is in assembly language, which assembler can understand

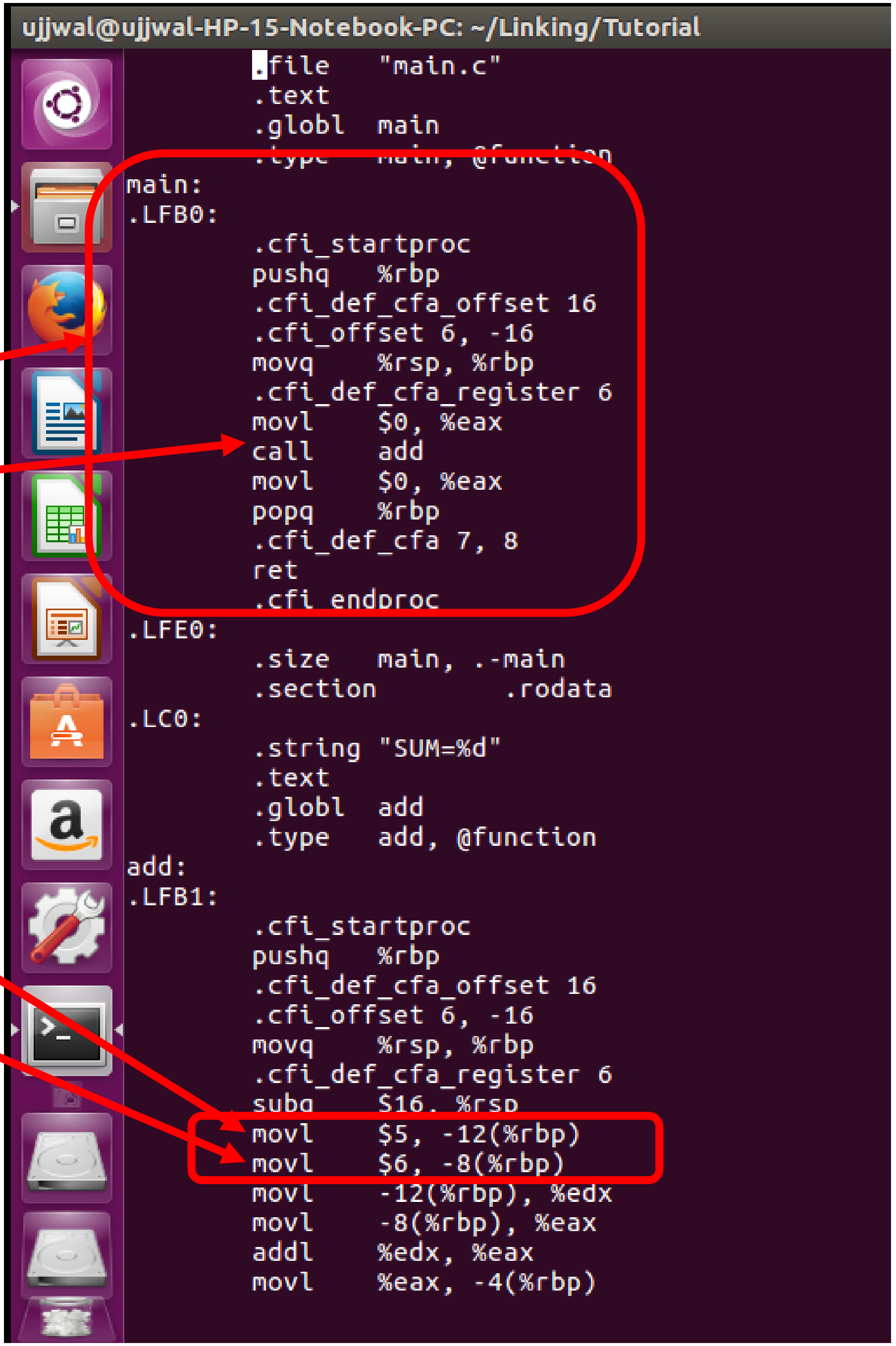
```
#include<stdio.h>

        int add();

        int main(){
              add();
        }

        int add(){
             int num1=5,
        num2=6, sum;
             sum=num1+num2;

        printf("SUM=%d",sum);
           return 0;
        }
```



ujjwal@ujjwal-HP-15-Notebook-PC: ~/Linking/Tutorial

```
        .file   "main.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0, %eax
        call    add
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .rodata
.LC0:
        .string "SUM=%d"
        .text
        .globl  add
        .type   add, @function
add:
.LFB1:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -12(%rbp)
        movl    $6, -8(%rbp)
        movl    -12(%rbp), %edx
        movl    -8(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
```

- Let’s see through this
- file using **$vi filename.s**
- The snapshot shows that it is in assembly language, which assembler can understand

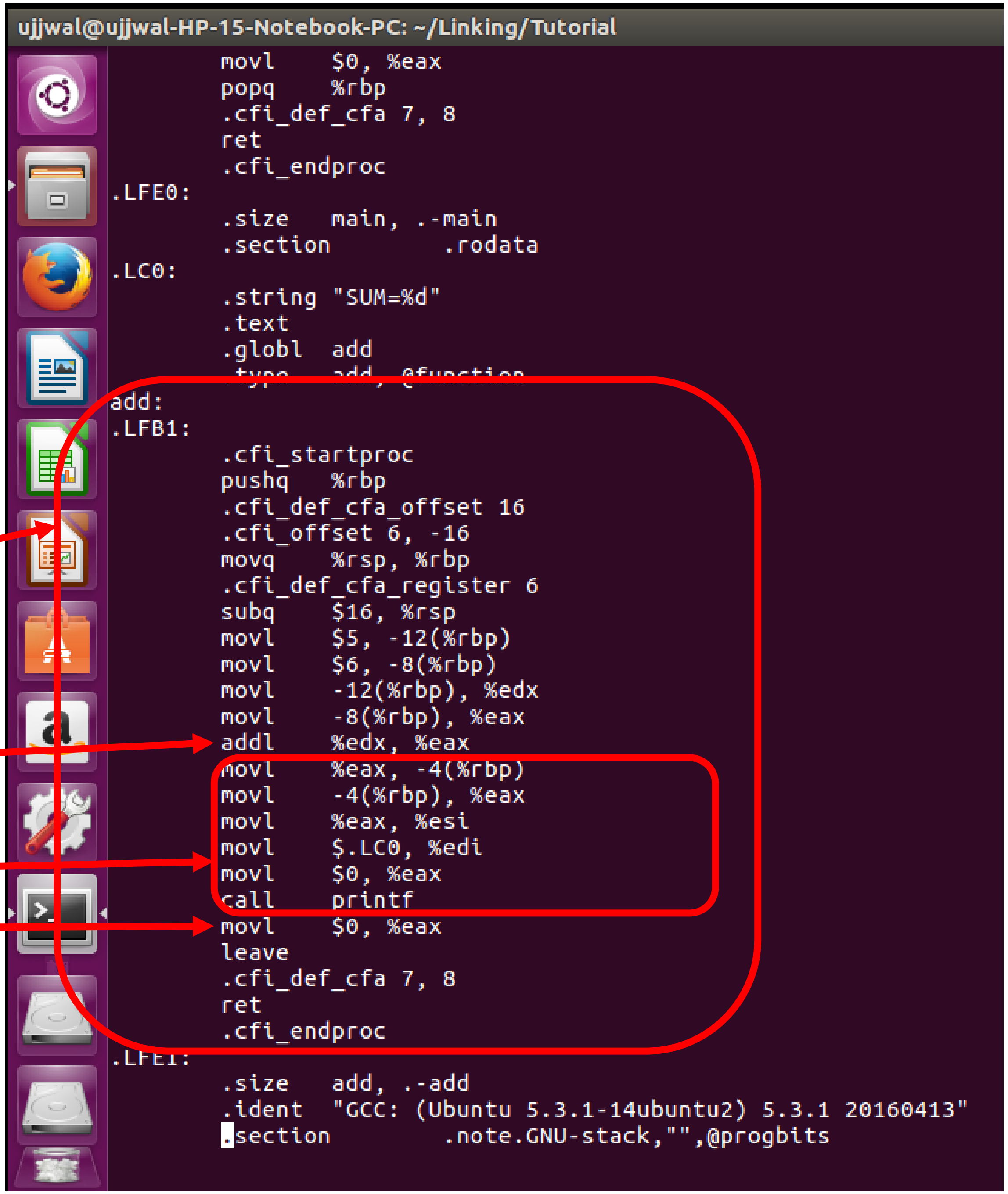
```
#include<stdio.h>

    int add();

    int main(){
        add();
    }

    int add(){
        int num1=5,
    num2=6, sum;
        sum=num1+num2;

    printf("SUM=%d",sum);
      return 0;
    }
```



ujjwal@ujjwal-HP-15-Notebook-PC: ~/Linking/Tutorial

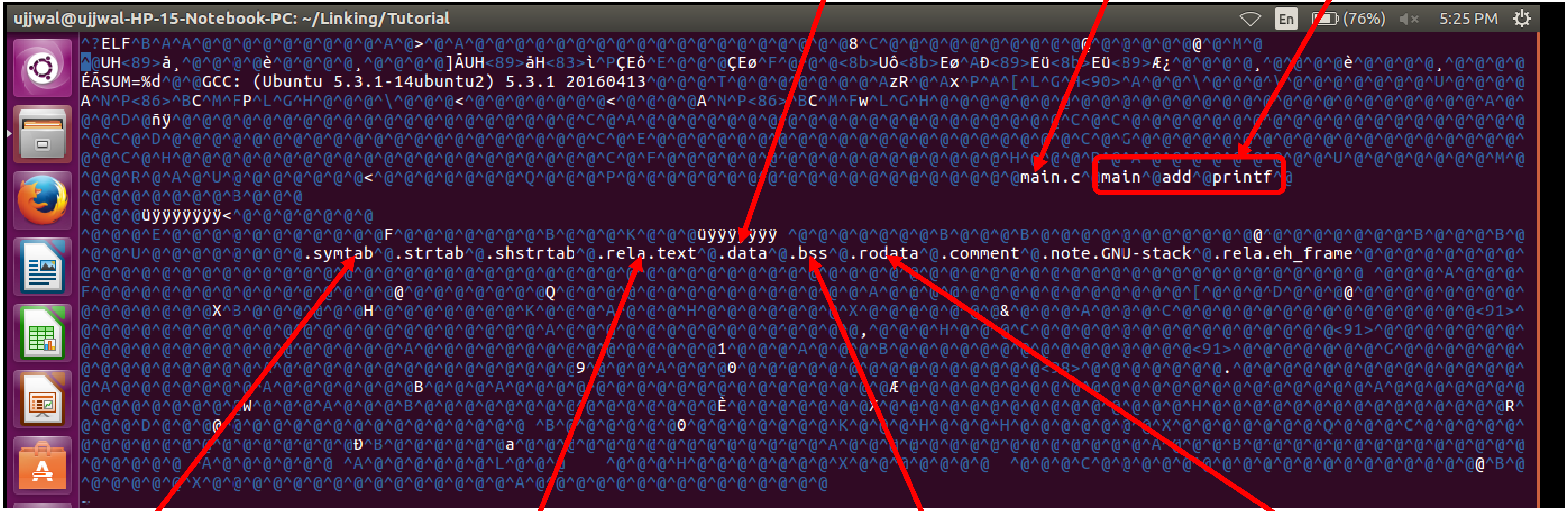
```
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .rodata
.LC0:
        .string "SUM=%d"
        .text
        .globl  add
        .type   add, @function
add:
.LFB1:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -12(%rbp)
        movl    $6, -8(%rbp)
        movl    -12(%rbp), %edx
        movl    -8(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   add, .-add
        .ident  "GCC: (Ubuntu 5.3.1-14ubuntu2) 5.3.1 20160413"
        .section        .note.GNU-stack,"",@progbits
```

Assembly

- In this phase the filename.s is taken as input and turned into **filename.o** by assembler
    - Translates main.s into a *relocatable object file* main.o
- This file contains machine level instructions
- Existing code is converted into machine language
- The function calls like printf() are not resolved

Contents of object file

- Let’s view this file using **$vi main.o**



```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@8^C^@^@^@^@^@^@^@^@^@@^@^@^@^@^@@^@^M^@
^@UH<89>å¸^@^@^@^@è^@^@^@^@¸^@^@^@^@]ÃUH<89>åH<83>ì^PÇEô^E^@^@^@ÇEø^F^@^@^@<8b>Uô<8b>Eø^AÐ<89>Eü<8b>Eü<89>Æ¿^@^@^@^@¸^@^@^@^@è^@^@^@^@¸^@^@^@^@
ÉÃSUM=%d^@^@GCC: (Ubuntu 5.3.1-14ubuntu2) 5.3.1 20160413^@^@^@^T^@^@^@^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^H<90>^A^@^@^\^@^@^@^\^@^@^@^@^@^@^@^@^@^U^@^@^@
A^N^P<86>^BC^M^FP^L^G^H^@^@^@^\^@^@^@<^@^@^@^@^@^@<^@^@^@^@A^N^P<86>^BC^M^Fw^L^G^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^
@^@^D^@ñÿ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^C^@^D^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^E^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^G^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^C^@^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^F^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^H^@^@^@^R^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^U^@^@^@^@^@^@^@^M^@
^@^@^R^@^A^@^U^@^@^@^@^@^@<^@^@^@^@^@^@^Q^@^@^@^P^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@main.c^@main^@add^@printf^@
^@^@^@^@^@^@^@^B^@^@^@
^@^@^@üÿÿÿÿÿÿÿ<^@^@^@^@^@^@
^@^@^@^E^@^@^@^@^@^@^@^@^@F^@^@^@^@^@^@^B^@^@^@^K^@^@^@üÿÿÿÿÿÿÿ ^@^@^@^@^@^@^B^@^@^@^B^@^@^@^@^@^@^@^@@^@^@^@^@^@^B^@^@^@^B^@
^@^@^U^@^@^@^@^@^@^@.symtab^@.strtab^@.shstrtab^@.rela.text^@.data^@.bss^@.rodata^@.comment^@.note.GNU-stack^@.rela.eh_frame^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^@^@^@^A^@^@^@^
F^@^@^@^@^@^@^@^@^@^@^@^@^@^@@^@^@^@^@^@^@Q^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^[^@^@^@^D^@^@^@@^@^@^@^@^@^@^@^
@^@^@^@^@^@X^B^@^@^@^@^@^@H^@^@^@^@^@^@^@^K^@^@^@^A^@^@^@^H^@^@^@^@^@^@^@^@^@^X^@^@^@^@^@^@^@&^@^@^@^A^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<91>^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@,^@^@^@^H^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<91>^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@1^@^@^@^A^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<91>^@^@^@^@^@^@^G^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@9^@^@^@^A^@^@^@0^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<98>^@^@^@^@^@^@^@.^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^A^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@B^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Æ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@
^@^@^@^@^@^@^@W^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@È^@^@^@^@^@^@^@X^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@R^
@^@^@D^@^@^@@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^B^@^@^@^@^@^@0^@^@^@^@^@^@^@^K^@^@^@^H^@^@^@^H^@^@^@^@^@^@^@X^@^@^@^@^@^@^Q^@^@^@^C^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@Ð^B^@^@^@^@^@^@a^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@ ^A^@^@^@^@^@^@ ^A^@^@^@^@^@^@^L^@^@^@    ^@^@^@^H^@^@^@^@^@^@^@^X^@^@^@^@^@^@^@  ^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@@^B^@
^@^@^@^@^@X^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
~
```

Data segment

Object file header

List of function

.sysmtab :
entry for symbols

.rel.text : locations in .text section
need to modified when linker combine
 it in other object file

.bss :
 Uninitialized global variables

.rodata :
read-only data

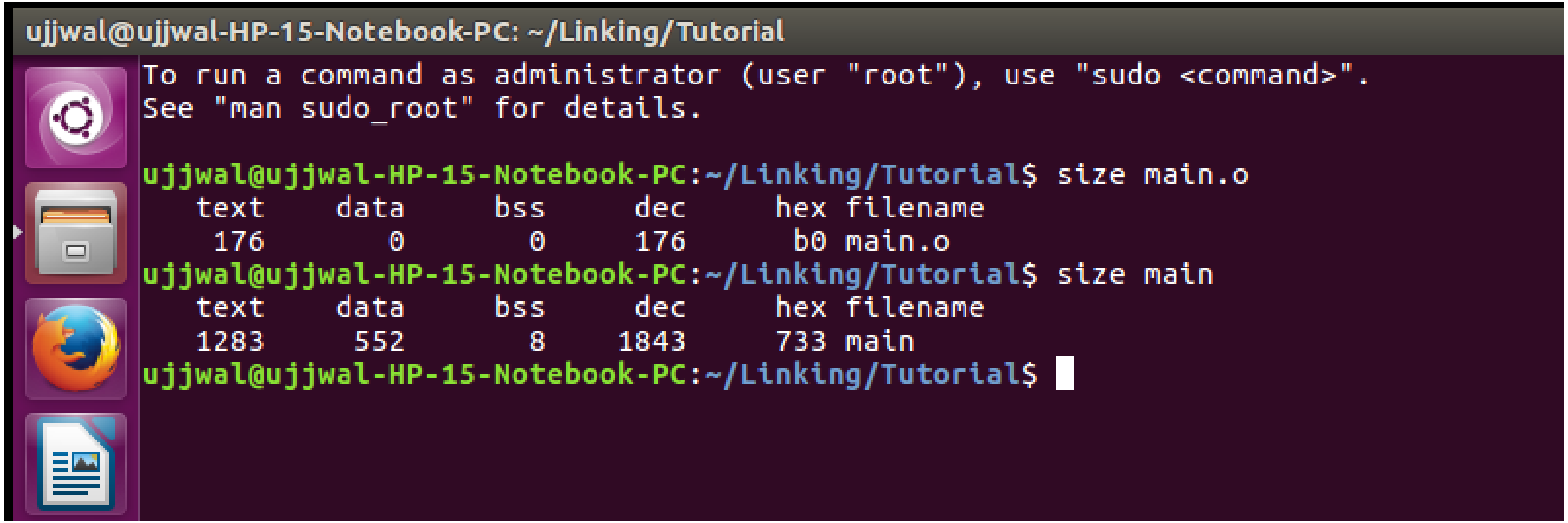
Linking

- In this phase, linking of all function calls with their definitions are done
  - it runs the linker program ld, which combines main.o, along with the necessary system object files, to create the *executable object file*
- Linker has the information where all these functions are defined
- It adds some extra code to the program, which is required when the program starts and ends
  - For example, there is a code which is required for setting up the environment like passing command line arguments

Linking (contd..)

- This task can be easily verified by using **$size filename.o** and **$size filename**.
- Through these commands, we know that how output file increases from an object file to an executable file. This is because of the extra code that linker adds with our program

Note : GCC by default does dynamic linking, so printf() is dynamically linked



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ujjwal@ujjwal-HP-15-Notebook-PC:~/Linking/Tutorial$ size main.o
   text    data     bss     dec     hex filename
    176       0       0     176      b0 main.o
ujjwal@ujjwal-HP-15-Notebook-PC:~/Linking/Tutorial$ size main
   text    data     bss     dec     hex filename
   1283     552       8    1843     733 main
ujjwal@ujjwal-HP-15-Notebook-PC:~/Linking/Tutorial$ 
```

Intermediate files generation

With single command

$gcc –Wall –save-temps filename.c –o filename

- “gcc” : Invokes the C compiler
- “-Wall” : gcc flag that enables all warnings. -W stands for warning, and we are passing “all” to -W.
- “-save-temps”: flag instructs compiler to store the temporary intermediate files used by the gcc compiler in the current directory
- “filename.c”: Input C program
- “-o filename”: Instruct C compiler to create the C executable as filename. If you don’t specify -o, by default C compiler will create the executable with name a.out

Intermediate files generation (contd..)

With multiple commands

- cpp –O2 main.c main.i
- gcc –S main.c
- as -o main.o main.s
- gcc –o p main.c

Intermediate files generation (contd..)

Consider two source files, main.c and add.c

main.c

```
#include<stdio.h>

int add();

int main(){
      add();
}
```

add.c

```
int add(){
     int num1=5, num2=6, sum;

     sum=num1+num2;
   printf("SUM=%d",sum);
   return 0;
}
```

Intermediate files generation (contd..)

- *gcc -O2 -g -o p main.c add.c*
- *./p*
- cpp [other arguments] main.c /tmp/main.i
  - translates the C source file main.c into an ASCII intermediate file main.i
- cc1 /tmp/main.i main.c -O2 [other arguments] -o /tmp/main.s
  - translates main.i into an ASCII assembly language file main.s
- as [other arguments] -o /tmp/main.o /tmp/main.s
  - translates main.s into a *relocatable object file* main.o
- ld -o p [system object files and args] /tmp/main.o /tmp/add.o
  - it runs the linker program ld, which combines main.o and add.o, along with the necessary system object files, to create the *executable object file* p

Intermediate files generation (contd..)

- bass> gcc -O2 -v -o p main.c add.c
- cpp main.c main.i
- cc1 main.i main.c -O2 -o main.s
- as -o main.o main.s
- <similar process for add.c>
- ld -o p main.o add.o
- bass>