**Operating System and Design (19CS2106A)**

**Advanced Lab- 1**

**Xv6 design, implementation, and customization.**

1. **add-a-user-program-to-xv6**

Clone xv6 in putty using the following link

git clone git://github.com/mit-pdos/xv6-public.git xv6

nano filename.c

Add a program and exit from nano editor

The Makefile needs to be edited to make our program available for the xv6 source code for compilation.

nano Makefile

Add filename in Makefile in uprogs and extras

make clean

make

make qemu-nox

1. **printf.c**

**#include "types.h"**

**#include "stat.h"**

**#include "user.h"**

**static void**

**putc(int fd, char c)**

**{**

**write(fd, &c, 1);**

**}**

**static void**

**printint(int fd, int xx, int base, int sgn)**

**{**

**static char digits[] = "0123456789ABCDEF";**

**char buf[16];**

**int i, neg;**

**uint x;**

**neg = 0;**

**if(sgn && xx < 0){**

**neg = 1;**

**x = -xx;**

**} else {**

**x = xx;**

**}**

**i = 0;**

**do{**

**buf[i++] = digits[x % base];**

**}while((x /= base) != 0);**

**if(neg)**

**buf[i++] = '-';**

**while(--i >= 0)**

**putc(fd, buf[i]);**

**}**

**// Print to the given fd. Only understands %d, %x, %p, %s.**

**void**

**printf(int fd, const char \*fmt, ...)**

**{**

**char \*s;**

**int c, i, state;**

**uint \*ap;**

**state = 0;**

**ap = (uint\*)(void\*)&fmt + 1;**

**for(i = 0; fmt[i]; i++){**

**c = fmt[i] & 0xff;**

**if(state == 0){**

**if(c == '%'){**

**state = '%';**

**} else {**

**putc(fd, c);**

**}**

**} else if(state == '%'){**

**if(c == 'd'){**

**printint(fd, \*ap, 10, 1);**

**ap++;**

**} else if(c == 'x' || c == 'p'){**

**printint(fd, \*ap, 16, 0);**

**ap++;**

**} else if(c == 's'){**

**s = (char\*)\*ap;**

**ap++;**

**if(s == 0)**

**s = "(null)";**

**while(\*s != 0){**

**putc(fd, \*s);**

**s++;**

**}**

**} else if(c == 'c'){**

**putc(fd, \*ap);**

**ap++;**

**} else if(c == '%'){**

**putc(fd, c);**

**} else {**

**// Unknown % sequence. Print it to draw attention.**

**putc(fd, '%');**

**putc(fd, c);**

**}**

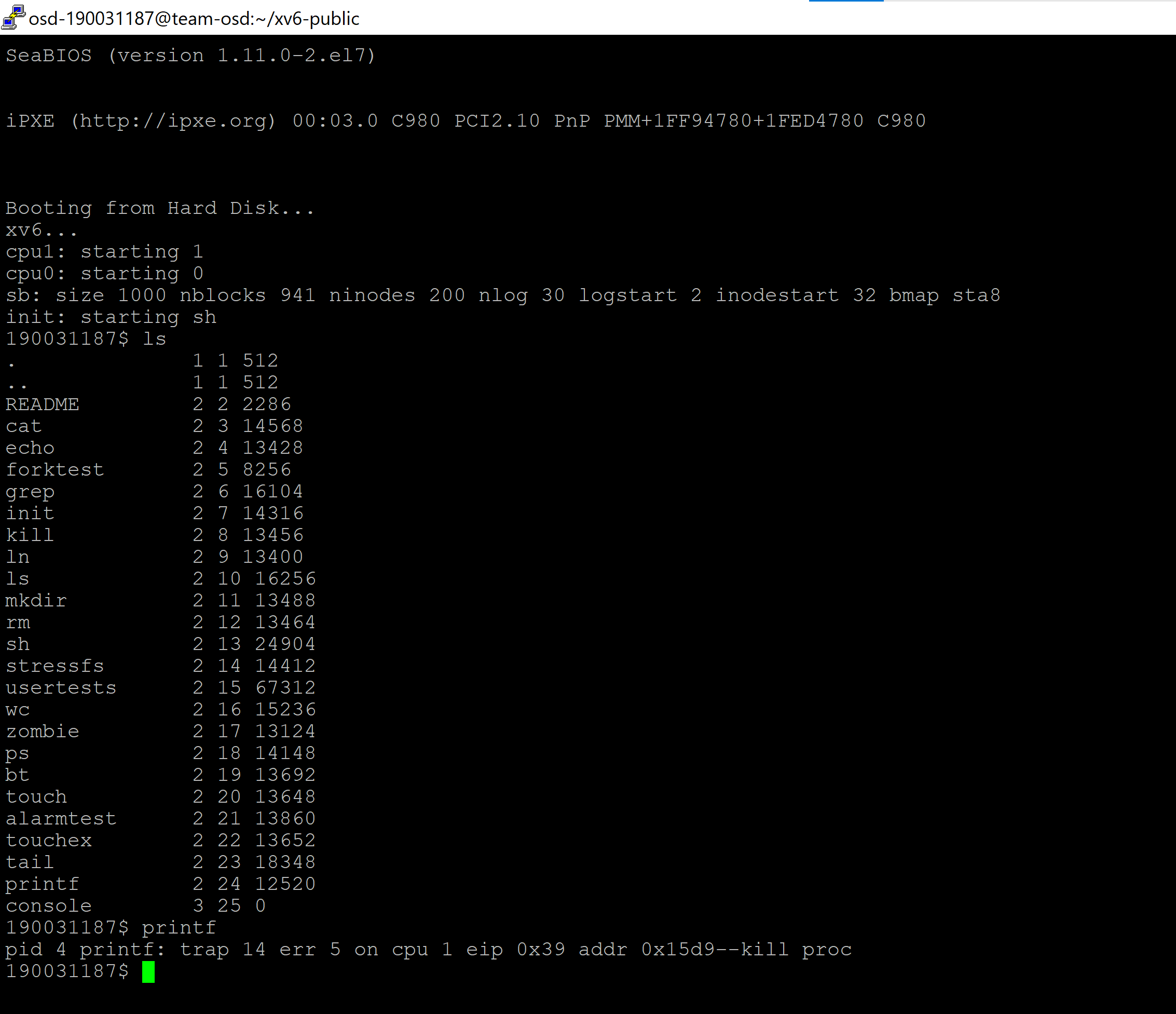
**state = 0;**

**}**

**}**

**}**

**OUTPUT**

****

**UNIX system programming**

**Compile-Link in GCC**

Compile any program u like in gcc

nano filename.c

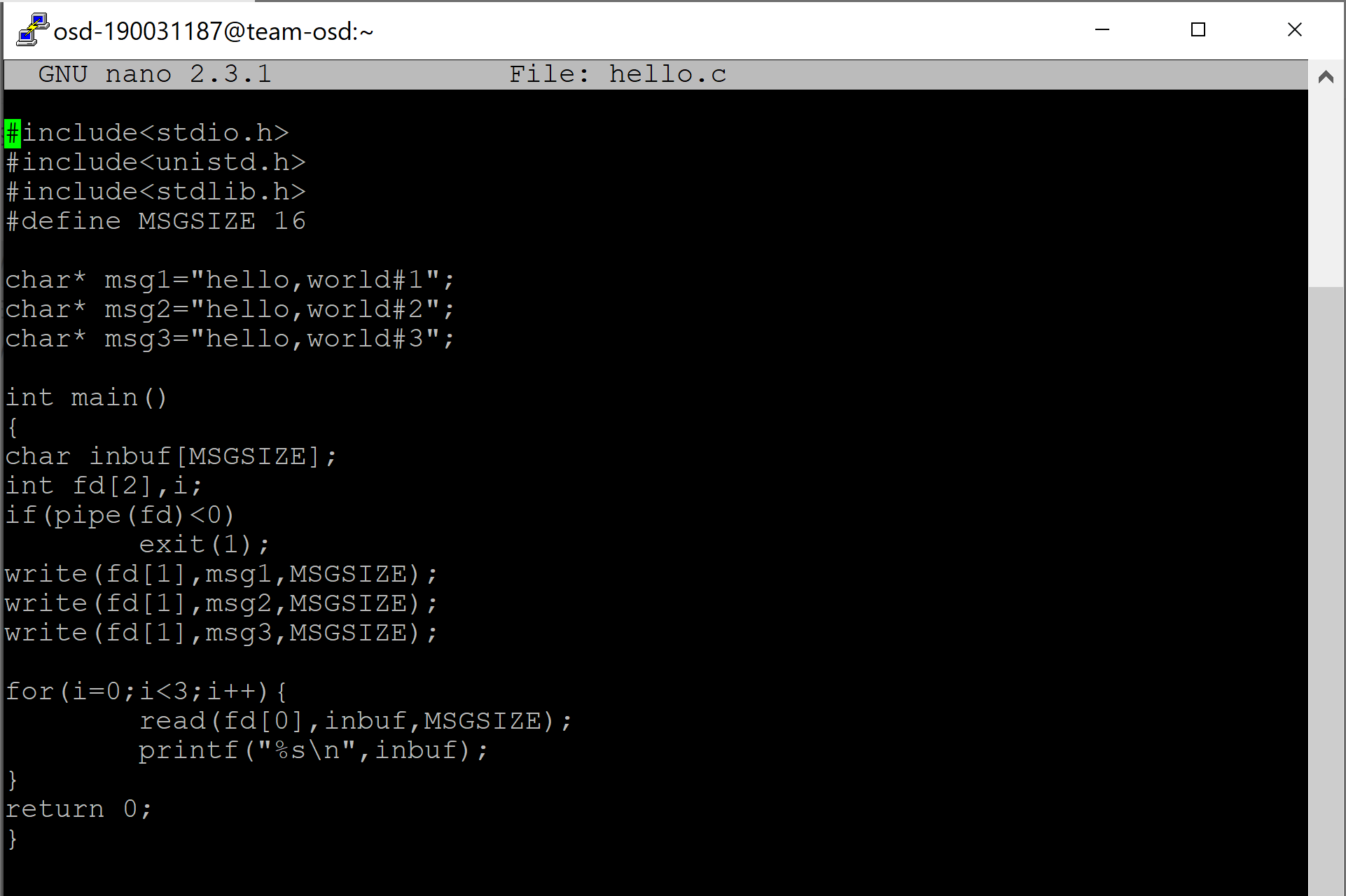
Add code and exit

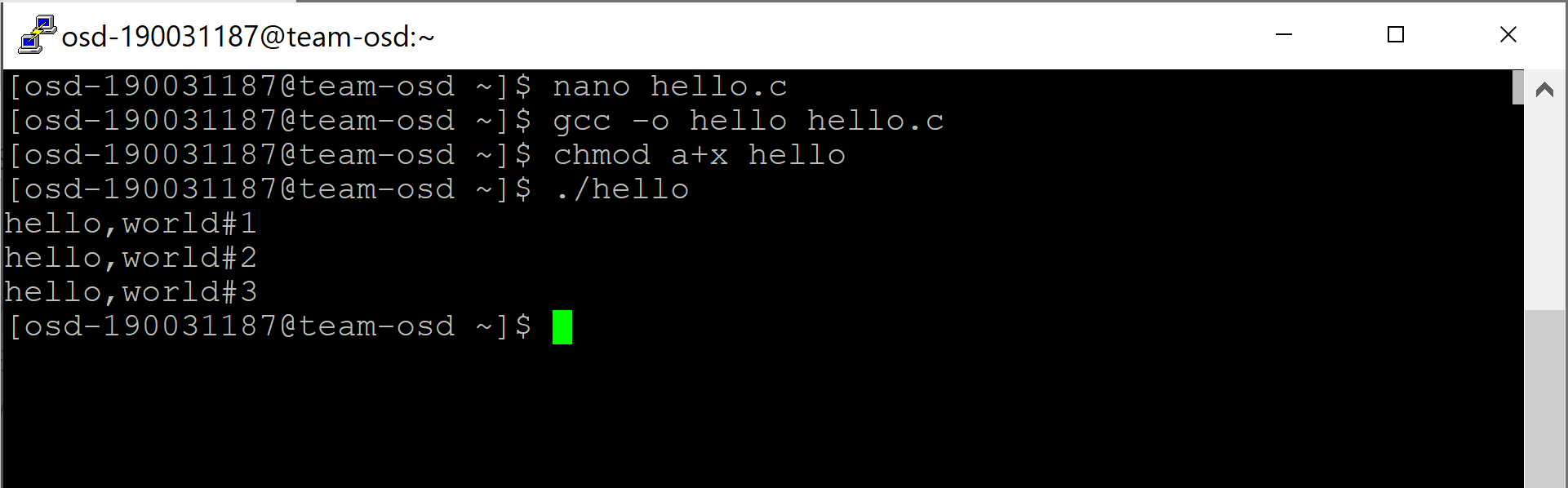
gcc filename.c

./a.out

Then link the code with another filename which doesnot exist in system using

link filename1.c filename2.c





**Static vs Dynamic Linking**

**Static Linking:**

When we click the .exe (executable) file of the program and it starts running, all the necessary contents of the binary file have been loaded into the process’s virtual address space. However, most programs also need to run functions from the system libraries, and these library functions also need to be loaded. In the simplest case, the necessary library functions are embedded directly in the program’s executable binary file. Such a program is statically linked to its libraries, and statically linked executable codes can commence running as soon as they are loaded.

**Dynamic Linking:**

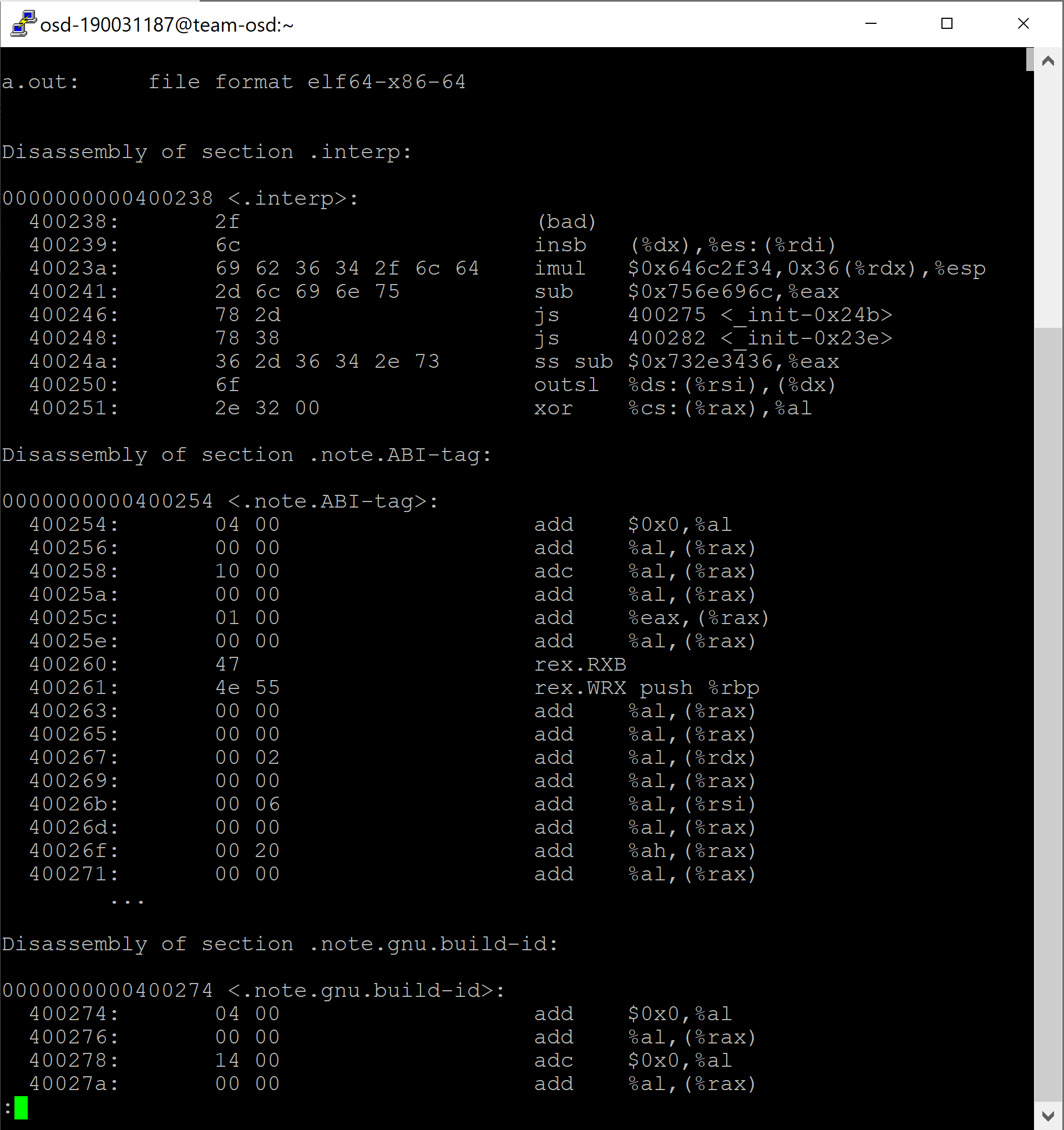
Every dynamically linked program contains a small, statically linked function that is called when the program starts. This static function only maps the link library into memory and runs the code that the function contains. The link library determines what are all the dynamic libraries which the program requires along with the names of the variables and functions needed from those libraries by reading the information contained in sections of the library. After which it maps the libraries into the middle of virtual memory and resolves the references to the symbols contained in those libraries. We don’t know where in the memory these shared libraries are actually mapped: They are compiled into positionindependent code (PIC), that can run at any address in memory

**Contents of a.out File**

not so sure for this answer

If I’m not wrong use

objdump -i filename to see contents of a.out file

****

**Program Execution and termination.**

nano filename.c

Add code and exit

gcc filename.c

./a.out

We have various ways to terminate a process

Cntrl+z (or)

killall gcc (or)

kill -s SIGINT %1

exit()

