


## (2025 OS) 1. Environment Setup

During this course, we will engage in multiple course projects to acquire practical knowledge and experience with the diverse OS features. The reference environment is [Ubuntu 22.04.2 LTS](#).

- [A. Install Ubuntu on VM](#)
  - [B. Install the development tools](#)
  - [C. Build and trace the Hello World program](#)
  - [<Exercise and Submission Checklist>](#)
  - [Reference](#)
- 

### A. Install Ubuntu on VM

If you are not running Ubuntu 22.04 natively on your computer, you may set up a VM and install Ubuntu on the VM. The following link points to instructions on how to install Ubuntu on a Hyper-V VM on Windows for your reference.

 [Install Ubuntu as Hyper-V Generation 2 Virtual Machine - .matrixpost.net](https://blog.matrixpost.net/install-ubuntu-as-hyper-v-generation-2-virtual-machine/) ( <https://blog.matrixpost.net/install-ubuntu-as-hyper-v-generation-2-virtual-machine/> )

The screen resolution can be changed via the following Windows Powershell command.

```
1 set-vmvideo Ubuntu -horizontalresolution:1920 -verticalresolution:1080 -resolutiontype single
```

It is fine to choose other VM platforms.

### B. Install the development tools

After you install Ubuntu 22.04, you need to install the following software packages for the project.

```
1 sudo apt-get install git
2 sudo apt-get install g++
```

## C. Build and trace the Hello World program

Open your editor and create the following C++ program

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello World\n");
6     return 0;
7 }
```

(main.cpp)

Compile the program and start gdb

```
1 g++ -g -o main main.cpp
2 gdb
```

In the GDB Window, type the following command and check the output as demonstrated by the screenshot below

1. Load the main executable => **'file main'**
2. Show the source code symbols embedded in the executable with **'list'**
3. Set a breakpoint on the print call at Line 5 with **'b 5'**
4. Start running the program with **'r'**
5. Set breakpoints on the future system call invocations with **'catch syscall'**
6. Continue program execution with **'c'**. GDB reports that it catches a system call in **\_\_GI\_\_fstate64**
7. Show the call stack with **'bt'**. It shows how the printf call (Line 5 in main.cpp) leads to **\_\_GI\_\_fstate64**
8. Use **x/20i** to disassemble 20 instructions from the beginning of **\_\_GI\_\_fstate64**.
9. We see that the fourth instruction of **\_\_GI\_\_fstate64** is the **syscall** instruction.

```

(gdb) file main
Reading symbols from main...
(gdb) list
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello World\n");
6      return 0;
7  }
(gdb) b 5
Breakpoint 1 at 0x1151: file main.cpp, line 5.
(gdb) r
Starting program: /home/hank/os_2023/p1/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:5
5      printf("Hello World\n");
(gdb) catch syscall
Catchpoint 2 (any syscall)
(gdb) c
Continuing

Catchpoint 2 (call to syscall newfstatat), __GI___fstatat64 (fd=1, file=0x7fffff7f5b46f "", buf=0x7ffffffffffdd70, flag=4096) at ../sysdeps/unix/sysv/linux/fstatat64.c:166
166      ...
(gdb) bt
#0  __GI___fstatat64 (fd=1, file=0x7fffff7f5b46f "", buf=0x7ffffffffffdd70, flag=4096) at ../sysdeps/unix/sysv/linux/fstatat64.c:166
#1  0x00007fffff7e01bf3 in __GI__IO_file_doallocate (fp=0x7fffff7f9d780 <_IO_2_1_stdout_>) at ./libio/libioP.h:947
#2  0x00007fffff7e10d60 in __GI__IO_doallocbuf (fp=fp@entry=0x7fffff7f9d780 <_IO_2_1_stdout_>) at ./libio/libioP.h:947
#3  0x00007fffff7e0ffe0 in _IO_new_file_overflow (f=0x7fffff7f9d780 <_IO_2_1_stdout_>, ch=-1) at ./libio/fileops.c:744
#4  0x00007fffff7e0e755 in _IO_new_file_xsputn (n=11, data=<optimized out>, f=<optimized out>) at ./libio/libioP.h:947
#5  _IO_new_file_xsputn (f=0x7fffff7f9d780 <_IO_2_1_stdout_>, data=<optimized out>, n=11) at ./libio/fileops.c:1196
#6  0x00007fffff7e03f9c in __GI__IO_puts (str=0x555555556004 "Hello World") at ./libio/libioP.h:947
#7  0x0000555555555160 in main () at main.cpp:5
(gdb) x/20i __GI___fstatat64
__GI___fstatat64>: endbr64
0x7fffff7e96ee4 <__GI___fstatat64+4>: mov     %ecx,%r10d
0x7fffff7e96ee7 <__GI___fstatat64+7>: mov     $0x106,%eax
0x7fffff7e96eec <__GI___fstatat64+12>: syscall
=> 0x7fffff7e96eee <__GI___fstatat64+14>: jmp     0xf7ffff000,%eax
0x7fffff7e96ef3 <__GI___fstatat64+19>: ja      0x7fffff7e96f00 <__GI___fstatat64+32>
0x7fffff7e96ef5 <__GI___fstatat64+21>: xor     %eax,%eax
0x7fffff7e96ef7 <__GI___fstatat64+23>: ret
0x7fffff7e96ef8 <__GI___fstatat64+24>: nopl    0x0(%rax,%rax,1)
0x7fffff7e96ef8 <__GI___fstatat64+24>: nopl    0x0(%rax,%rax,1)
0x7fffff7e96f00 <__GI___fstatat64+32>: mov     0x104f09(%rip),%rdx      # 0x7fffff7f9be10
0x7fffff7e96f07 <__GI___fstatat64+39>: neg     %eax
0x7fffff7e96f09 <__GI___fstatat64+41>: mov     %eax,%fs:(%rdx)
0x7fffff7e96f0c <__GI___fstatat64+44>: mov     $0xffffffff,%eax
0x7fffff7e96f11 <__GI___fstatat64+49>: ret
0x7fffff7e96f12: cs nopl 0x0(%rax,%rax,1)
0x7fffff7e96f1c: nopl    0x0(%rax)
0x7fffff7e96f20 <statx_generic>: push    %rbx
0x7fffff7e96f21 <statx_generic+1>: sub     $0x1a0,%rsp
0x7fffff7e96f28 <statx_generic+8>: mov     %fs:0x28,%rax

```

## <Exercise and Submission Checklist>

Now, please find another library call (other than **printf**) and shows the call stack leading to the 'syscall' instruction.

Prepare a report in PDF format with the following items

A1. Description of your setup and your experiment steps.

A2. Results of your experiment. You can use screenshots, copies of the output messages from the tools, or recording a video (need to supply the URL link to your video and make sure it is

accessible by the TA)

A3. Your source code (a zipped file)

## Reference

<https://blog.xuite.net/yh96301/blog/63289807->

[VMware+Player+7.0%E4%B8%8B%E8%BC%89%E8%88%87%E5%AE%89%E8%A3%9D](#)

[Download and Install VMware Workstation Player/Pro \(16/15/14\) \(minitool.com\).](#)

[Installing Linux Ubuntu Version 20.04 in VMware Workstation Player - Windows 11](#)

[Installation Guides \(dellwindowsreinstallationguide.com\).](#)

<https://blog.xuite.net/yh96301/blog/341981056->

[VMware+Workstation+16+Player%E5%AE%89%E8%A3%9DUbuntu+20.04%3E](#)

[Intel® 64 and IA-32 Architectures Software Developer Manuals](#)