

**实 验 报 告**

**课程名称：** **操作系统**

**姓 名：** **方桂安**

**学 号：** **20354027**

**专业班级：** **2020级智能科学与技术**

**任课教师：** **吴贺俊**

2022 **年** 9 **月** 2 **日**

**实验报告成绩评定表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 评定项目 | 内 容 | 满 分 | 评 分 | 总 分 |
| 实验态度 | 态度端正、遵守纪律、出勤情况 | 10 |  |  |
| 实验过程 | 按要求完成算法设计、代码书写、注释清晰、运行结果正确 | 30 |  |
| 实验记录 | 展示讲解清楚、任务解决良好、实验结果准确 | 20 |  |
| 报告撰写 | 报告书写规范、内容条理清楚、表达准确规范、上交及时、无抄袭，抄袭记0分，提供报告供抄袭者扣分。 | 40 |  |
| **评语：**  **指导老师签字： 年 月 日** | | | | |

实验报告正文

**实验一 Linux的安装与使用**

1. **实验目的**
2. 掌握Linux环境下的命令操作，熟悉Linux操作系统的环境和使用，记录各种测试结果；
3. 了解LINUX系统的安装过程，记录安装流程和界面；
4. 搭建 ROS 环境，为后续实验做准备。
5. **实验内容**
   1. **任务描述**

1）Linux系统的安装

1. 调研、选择Linux版本
2. 搜索、下载Linux安装所需文件
3. 安装Linux

2）Linux基本操作命令

熟悉pwd、ls、mkdir、cd、cat、man、cp等命令的使用

3）编写程序

1. 编写能输出“Hello world!”问候语的C程序，并在终端中编译、执行。要求记录所使用的命令及结果。
2. 编写一个程序：显示信息“Time for Play!”，并能在后台运行一段时间（自定义）后，弹出信息提醒用户。要求记录所使用的命令及结果。（提示：使用sleep(s)函数）

4）安装ROS

* 1. 按照官方教程或者快捷脚本安装好对应版本的ROS；
  2. 运行ROS代码例程。
  3. **实验说明**

本次实验中我通过VMware和WSL2两种方式安装了Linux操作系统，磁盘管理、文件管理的命令熟悉和程序编写使用WSL2完成，ROS的安装与代码例程使用有可视化界面的VMware完成。

1. **实验记录**
   1. **实施步骤**
2. 使用VMware安装：

* 以管理员身份运行Vmware Workstation
* 载入并安装ubuntu-22.04.1-desktop-amd64.iso

1. 使用WSL2安装：

* 启用“适用于 Linux 的 Windows 子系统”可选功能
* 启用“虚拟机平台”可选功能
* 下载安装 Linux 内核更新包并将 WSL 2 设置为默认版本
* 使用 Microsoft Store，选择并安装偏好的 Linux 分发版

1. 在终端练习Linux的基本操作命令
2. 编写程序
3. 利用脚本安装ROS并运行代码例程
   1. **实验记录**
   2. 安装Linux

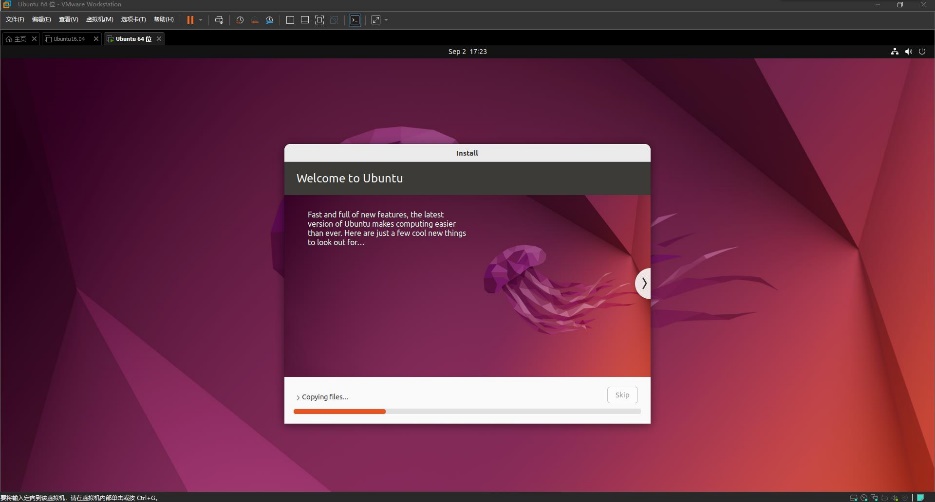
****

图1-VM安装过程

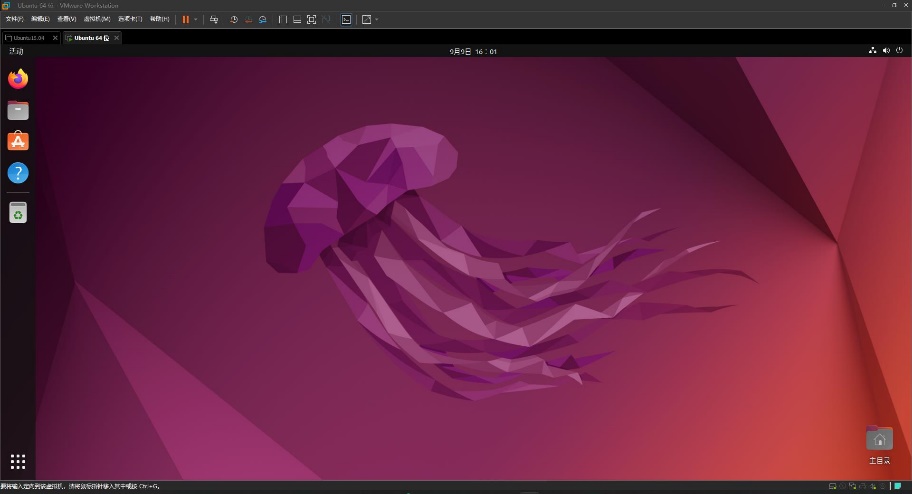


图2-安装成功

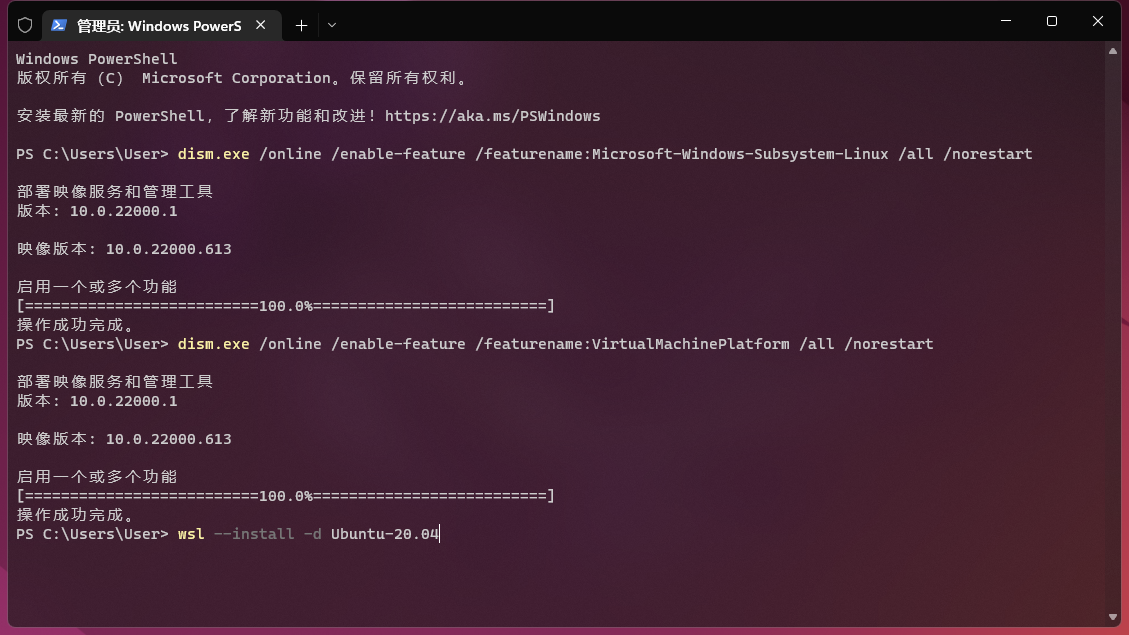


图3-WSL2安装过程

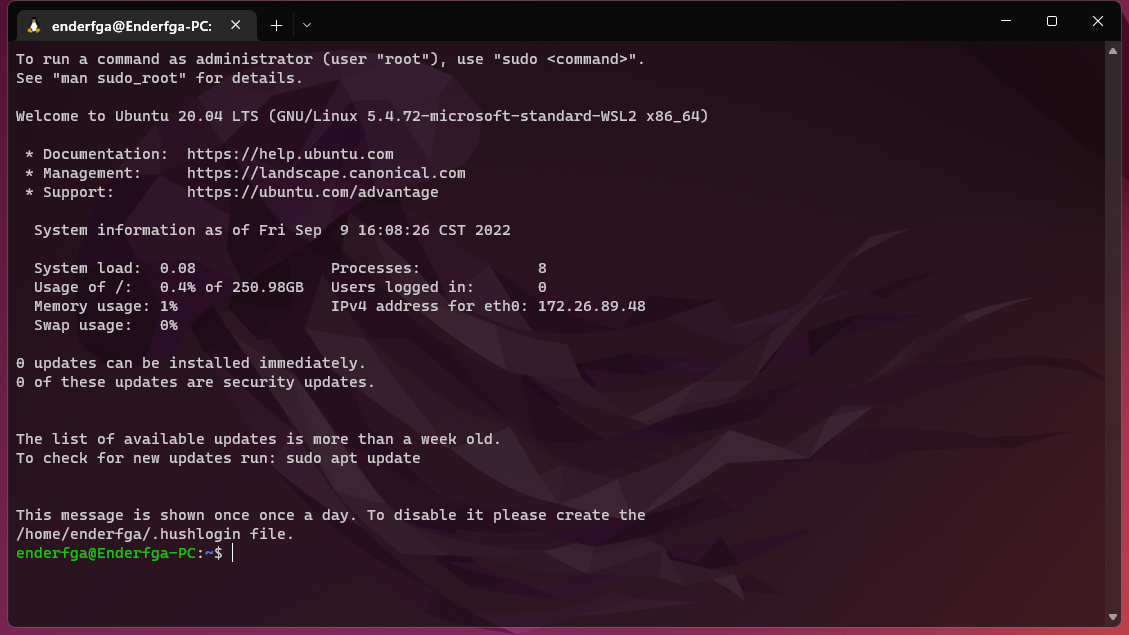


图4-WSL2安装成功

1. 熟悉操作命令

**pwd命令：**

英文全拼：print work directory，该命令用于显示工作目录，

执行 pwd 指令可立刻得知您目前所在的工作目录的绝对路径名称。

**ls命令：**

英文全拼：list files，该命令用于显示指定工作目录下之内容（列出目前工作目录所含之文件及子目录)。

参数说明：

* **-a** 显示所有文件及目录 (. 开头的隐藏文件也会列出)
* **-l** 除文件名称外，亦将文件型态、权限、拥有者、文件大小等资讯详细列出
* **-r** 将文件以相反次序显示(原定依英文字母次序)
* **-t** 将文件依建立时间之先后次序列出
* **-A** 同 -a ，但不列出 "." (目前目录) 及 ".." (父目录)
* **-F** 在列出的文件名称后加一符号；例如可执行档则加 "\*", 目录则加 "/"
* **-R** 若目录下有文件，则以下之文件亦皆依序列出

**mkdir命令：**

英文全拼：make directory，该命令用于创建目录。

参数说明：

* **-p** 确保目录名称存在，不存在的就建一个。

**cd命令：**

英文全拼：change directory，该命令用于切换当前工作目录。

其中 dirName 表示法可为绝对路径或相对路径。若目录名称省略，则变换至使用者的 home 目录 (也就是刚 login 时所在的目录)。

另外，~ 也表示为 home 目录 的意思， . 则是表示目前所在的目录， .. 则表示目前目录位置的上一层目录。

**cat命令：**

英文全拼：concatenate,该命令用于连接文件并打印到标准输出设备上。

参数说明：

* **-n 或 --number**：由 1 开始对所有输出的行数编号。
* **-b 或 --number-nonblank**：和 -n 相似，只不过对于空白行不编号。
* **-s 或 --squeeze-blank**：当遇到有连续两行以上的空白行，就代换为一行的空白行。
* **-v 或 --show-nonprinting**：使用 ^ 和 M- 符号，除了 LFD 和 TAB 之外。
* **-E 或 --show-ends**: 在每行结束处显示 $。
* **-T 或 --show-tabs**: 将 TAB 字符显示为 ^I。
* **-A, --show-all**：等价于 -vET。
* **-e：**等价于"-vE"选项；
* **-t：**等价于"-vT"选项；

**cp命令：**

英文全拼：copy file，该命令主要用于复制文件或目录。

参数说明：

* **-a**：此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于dpR参数组合。
* **-d**：复制时保留链接。这里所说的链接相当于 Windows 系统中的快捷方式。
* **-f**：覆盖已经存在的目标文件而不给出提示。
* **-i**：与 -f 选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答 y 时目标文件将被覆盖。
* **-p**：除复制文件的内容外，还把修改时间和访问权限也复制到新文件中。
* **-r**：若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件。
* **-l**：不复制文件，只是生成链接文件。

1. 编写程序

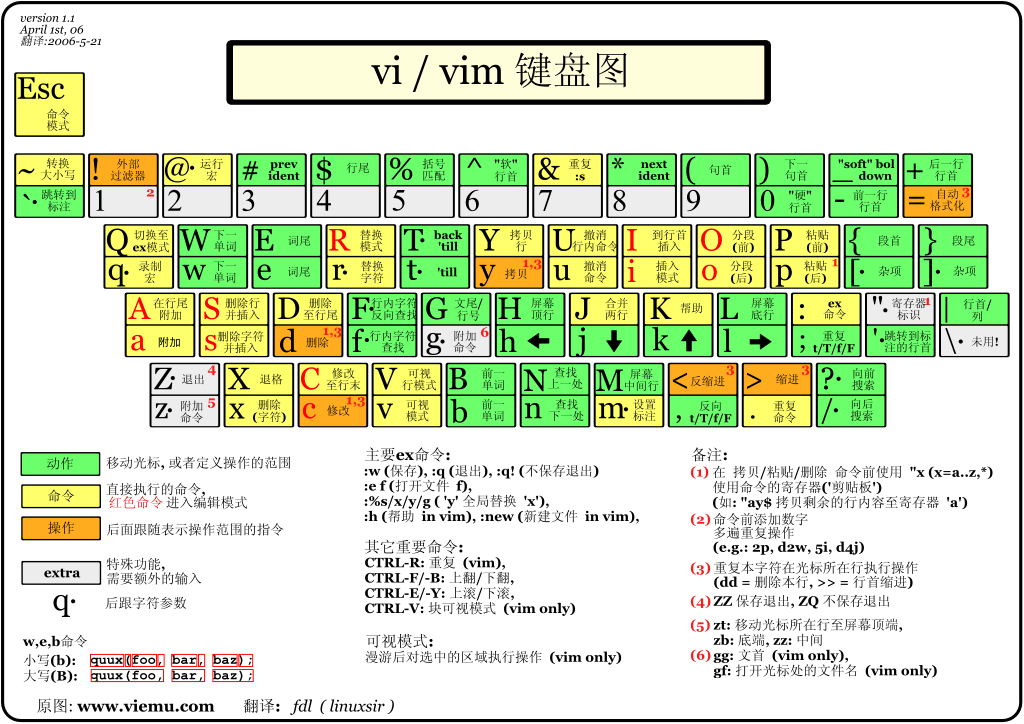


图5-vim 键盘图

使用vi文书编辑器来编写要求的程序，代码如下：

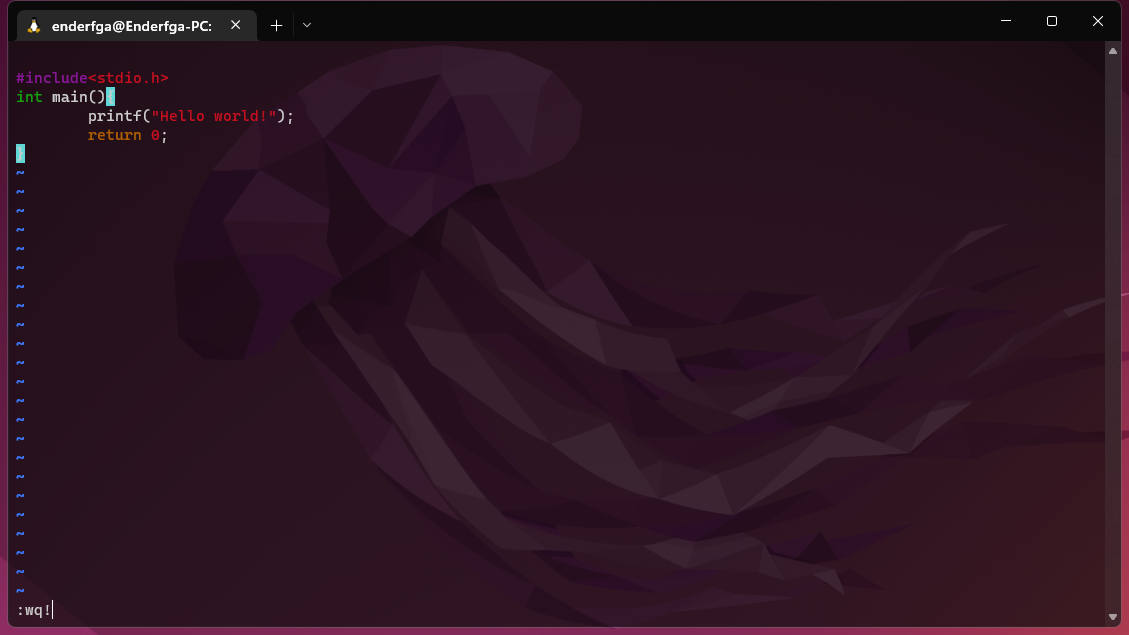


图6-编写程序一

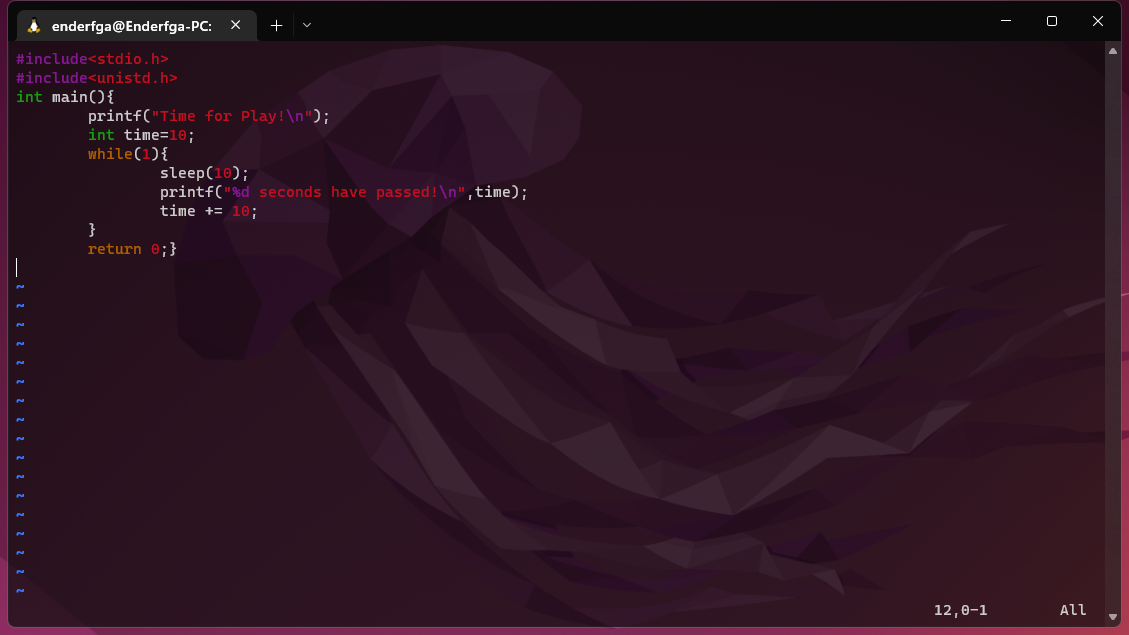


图7-编写程序二

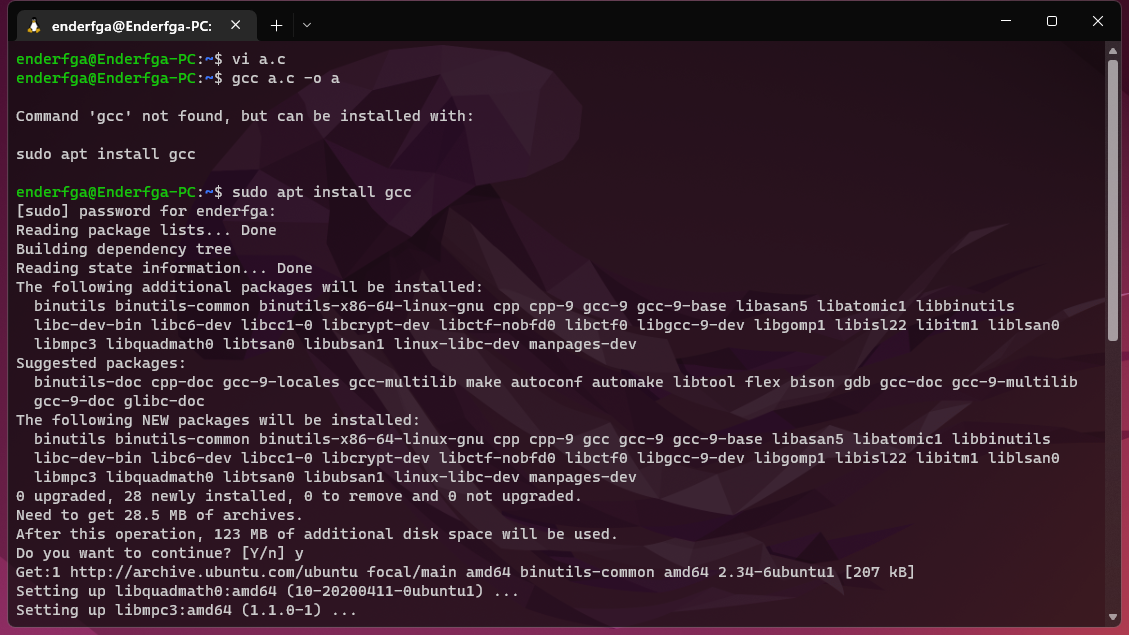


图8-安装gcc并编译

1. 安装ROS

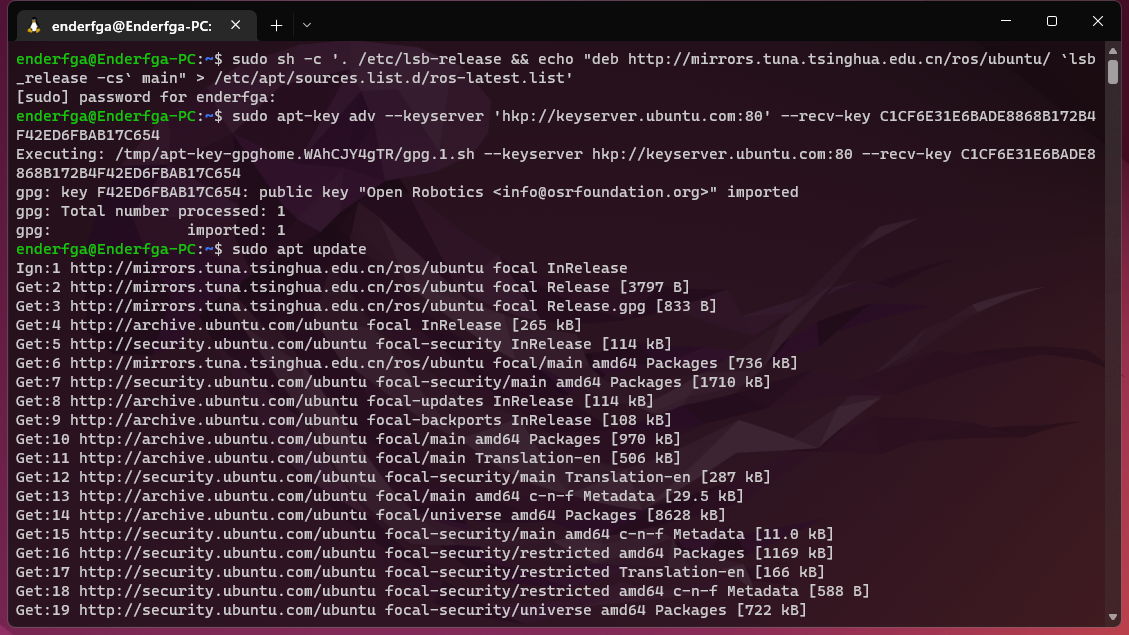


图9-使用官方教程手动安装

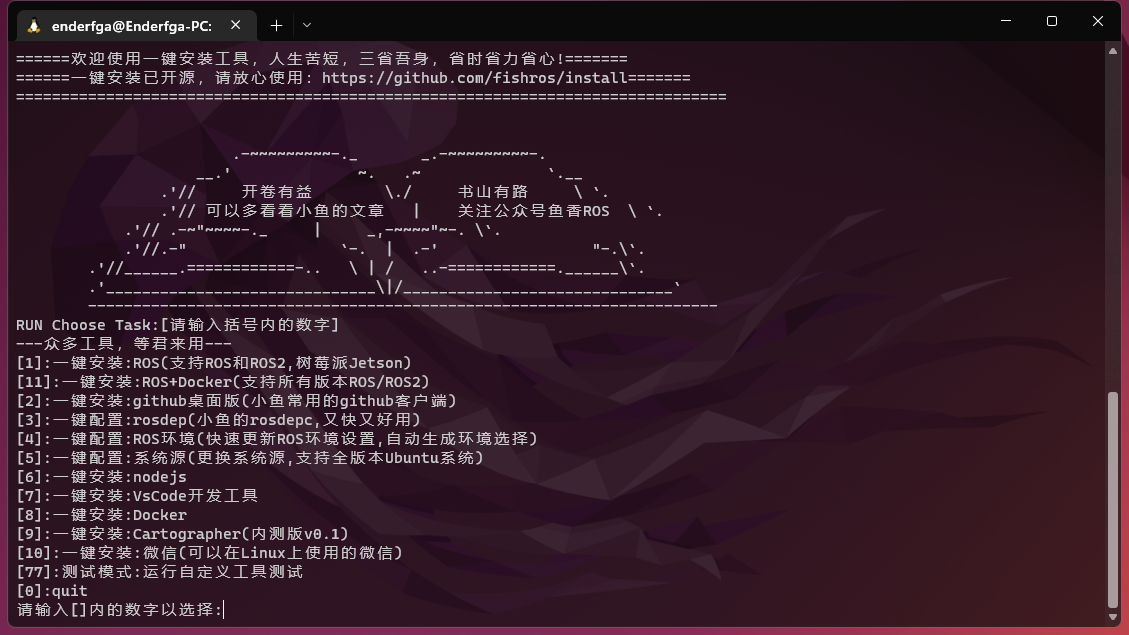


图10-使用一键安装脚本

* 1. **实验结果**

1. 命令熟悉

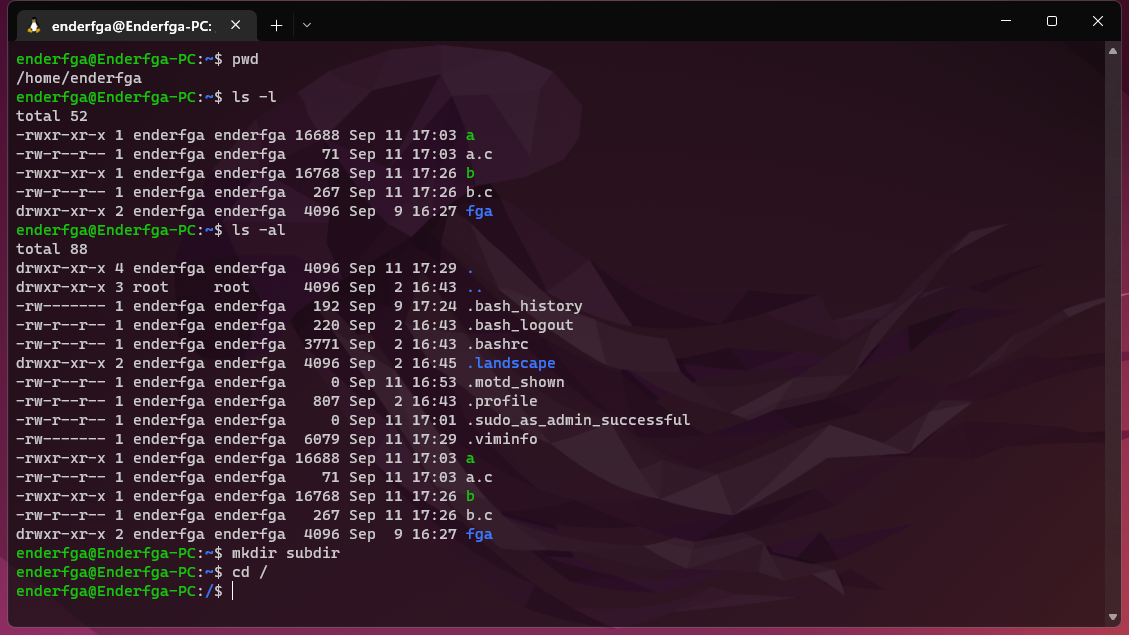


图11-基本操作命令结果一

如图所示，打开终端之后起始目录显示为“~”，即主目录，也就是当前登录用户的用户目录，故输入pwd的结果为/home/enderfga。

ls -l会显示当前目录下文件的详细信息，图中是我编写的两个c文件以及编译产生的文件等，ls -al则会将以“.”开头的隐藏文件也展示出来。

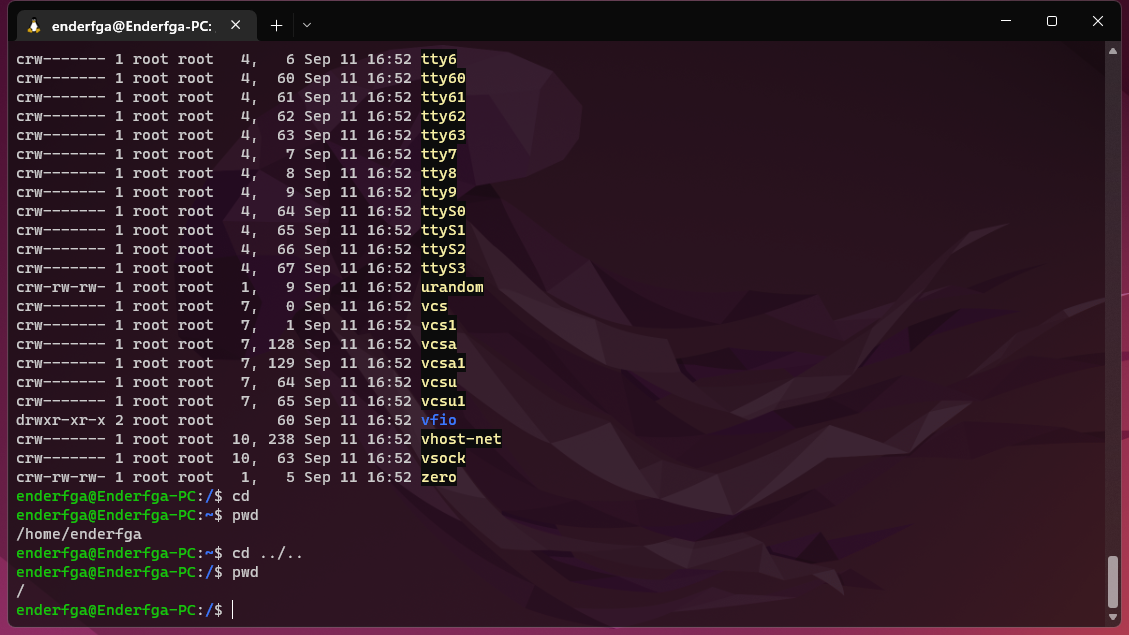


图12-基本操作命令结果二

直接使用不带参数的命令cd会改变目录至当前的用户目录，即“~”；

使用命令cd ../..会到上一级目录的上一级目录中，在此处也就是根目录，即“/”。

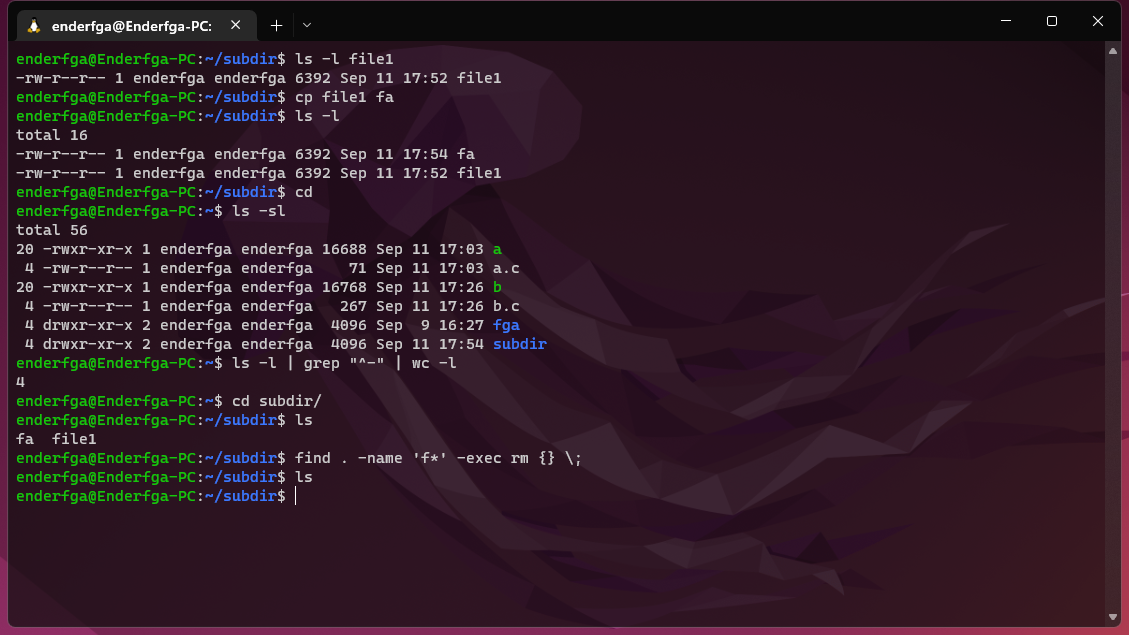


图13-文件操作命令结果

值得注意的是，输入命令ls -l后屏幕显示的第一列内容wsl2的结果和VMware的结果并不一样。

wsl2中显示的是文件大小，如果是文件，则表示该文件的大小，单位为字节；如果是目录，则表示该目录符所占的大小，并不表示该目录下所有文件的大小。

VMware的结果如图所示：

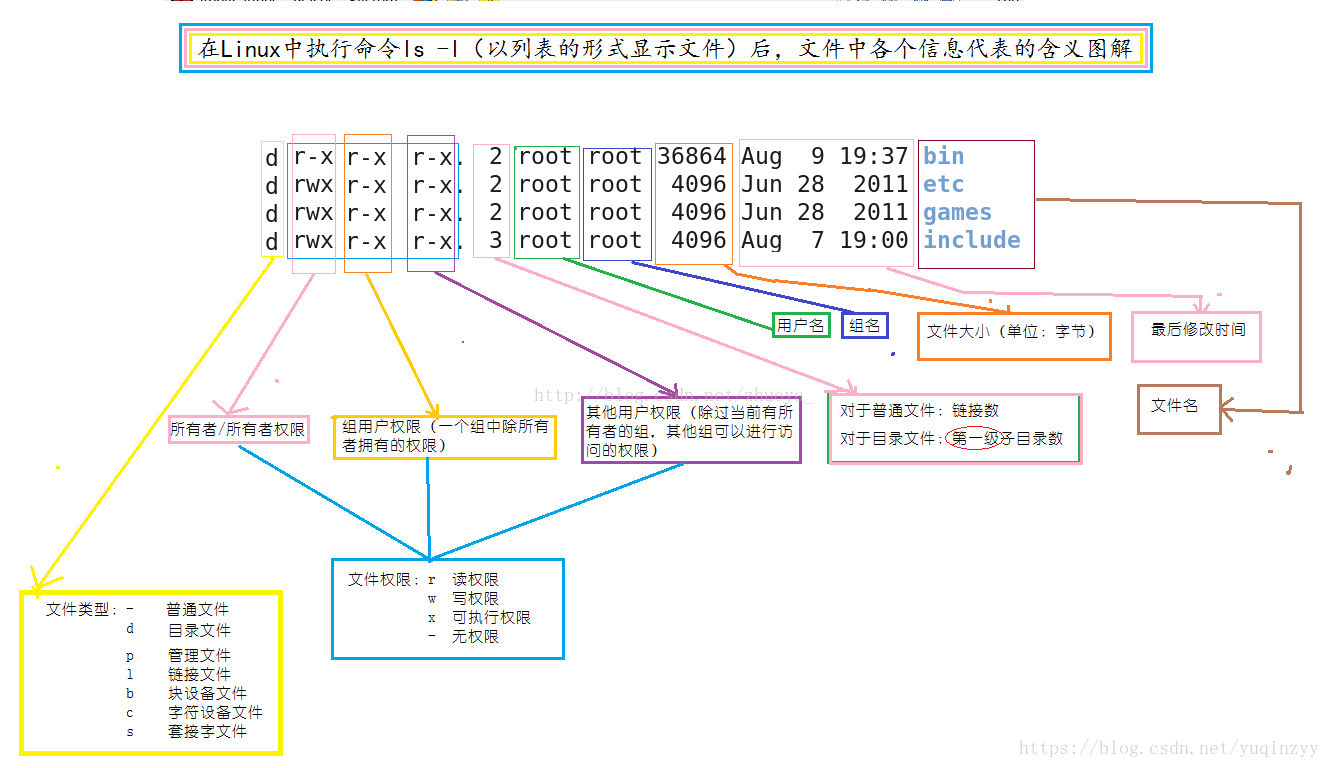


图14-ls -l命令解析

图中还记录了两个我搜索到的其他文件操作命令，分别是

“ls -l | grep "^-" | wc -l”：统计该目录下的文件个数；以及

“find . -name 'f\*' -exec rm {} \;”：搜寻含有特定文件名的文件并删除。

1. 程序编写

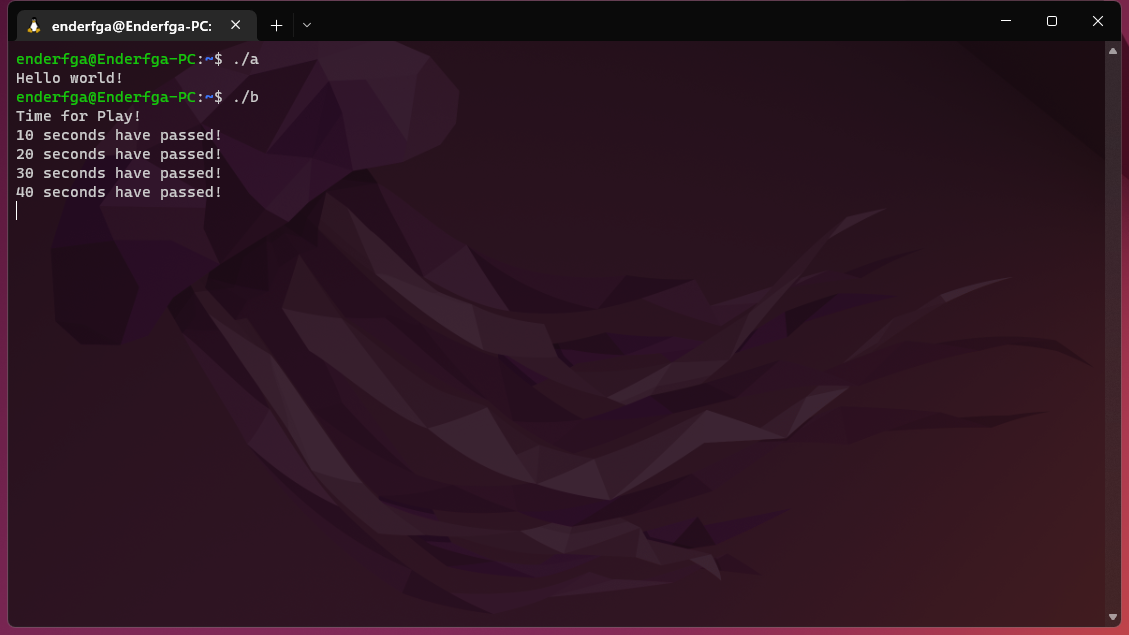


图15-C程序输出结果

根据题目要求，a.c编译的程序实现了输出“Hello world!”问候语；b.c编译的程序会在运行时显示“Time for Play!”，并能在后台运行，每10秒弹出信息提醒用户。

1. ROS测试与例程

值得注意的是，本次实验中我安装的是ROS2，故PPT中的命令不能生效。查阅资料之后，我使用ROS2对应的命令成功实现了用键盘控制小乌龟运动。



图16-ROS小乌龟测试

以下是我在过去的学习中使用ROS1和rviz可视化的A\*轨迹规划和Turtlebot2轨迹仿真，在此处记录。

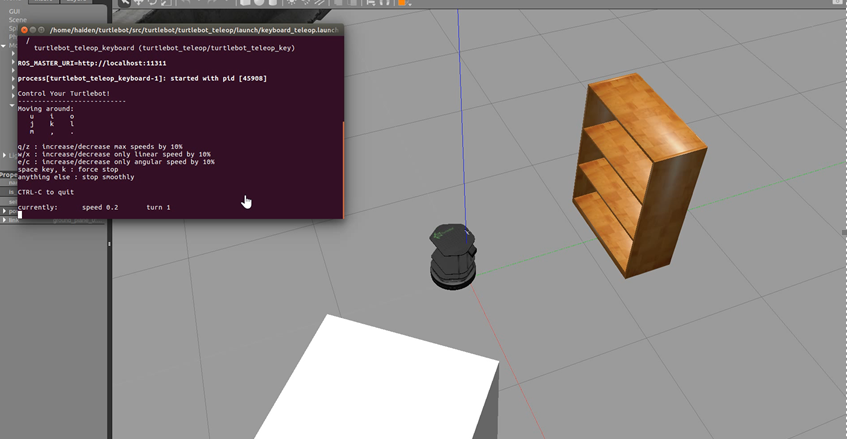
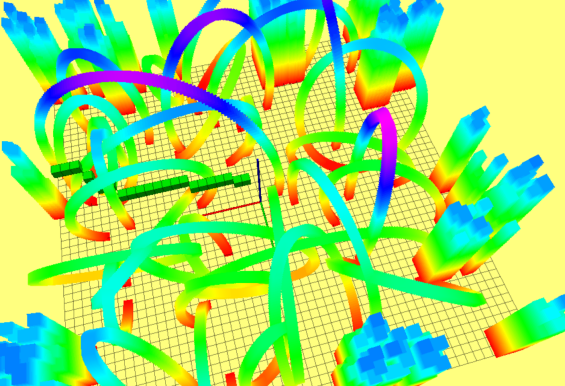


图17-轨迹规划与实机仿真

1. **总结与讨论**
   1. 在Linux中，如何设置前、后台命令和程序的执行？

在终端中如果对命令不加处理，那么命令会在前台运行；如果打算把命令放到后台运行，这个时候只需要在命令末尾加上&即可，此时终端返回的是[作业号] 进程号。

需要注意的是，如果程序在后台运行，那么它将无法接受用户的输入，但是其输出将显示在屏幕上（可能用户正在进行其他工作，突然冒出了错误输出），因此在后台执行的程序需是不需要人工干预的、输出被妥善处理（比如重定向）的程序。

有的时候在程序开始运行之后，想要将程序放在后台执行，这时需要按Ctrl+Z快捷键暂停程序，然后使用bg %作业号命令将其放入后台执行，也可以使用fg %作业号将程序从后台移到前台。

&是指在后台运行，但当用户推出(挂起)的时候，命令自动也跟着退出。而nohup命令（no hang up）是不挂断的运行，故二者常常结合起来使用，这样就能使命令永久的在后台执行，例如：

CUDA\_VISIBLE\_DEVICES=0,1 nohup python train.py > nohup.log 2>&1 &

即为使用2块GPU后台训练的常用python命令。

* 1. 你所使用的Linux系统的内核版本是多少？用什么命令查看内核版本？目前你所了解的各发行版本的情况如何？

可以使用uname -a来查看内核版本，本次实验使用的WSL2和VMware分别输出的结果是：

Linux Enderfga-PC 5.4.72-microsoft-standard-WSL2 #1 SMP Wed Oct 28 23:40:43 UTC 2020 x86\_64 x86\_64 x86\_64 GNU/Linux

Linux enderfga-virtual-machine 5.15.0-47-generic #51-Ubuntu SMP Thu Aug 11 07:51:15 UTC 2022 x86\_64 x86\_64 x86\_64 GNU/Linux

可以看出WSL2相较于虚拟机的差异还是挺大的，即使均为Ubuntu 20.04在内核版本上也有不同。

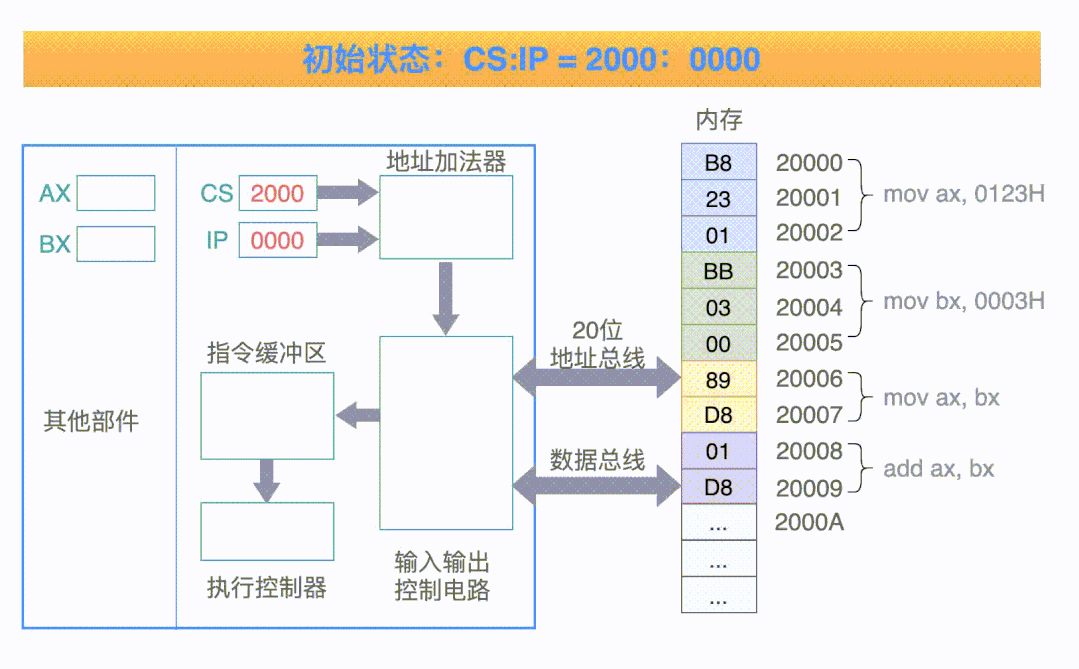
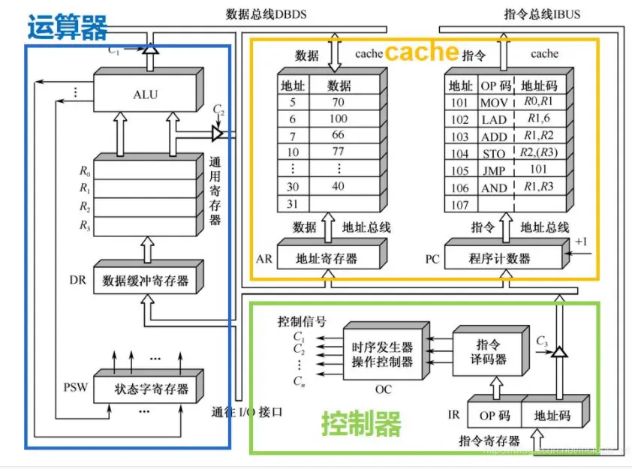
查阅资料得知，比较流行的发行版本有10个。Ubuntu，Linux Mint和PCLinuxOS是其中最易于使用的。想快速部署使用，就可以选择这几个。尤其对于新手，已经做到了和Windows类似的易用程度了；另一方面，Slackware Linux，Arch Linux和FreeBSD是更激进的发行版，更新比较频繁，所以需要有一定的基础；openSUSE，Fedora，Debian GNU/Linux和Mageia则是比较保守的发行版，稳定性是他们的特点，但是软件包都比较旧，很多桌面版本的新功能没法用；CentOS是一个企业级的发行版，适合那些喜欢稳定性，可靠性和软件长期支持的用户。

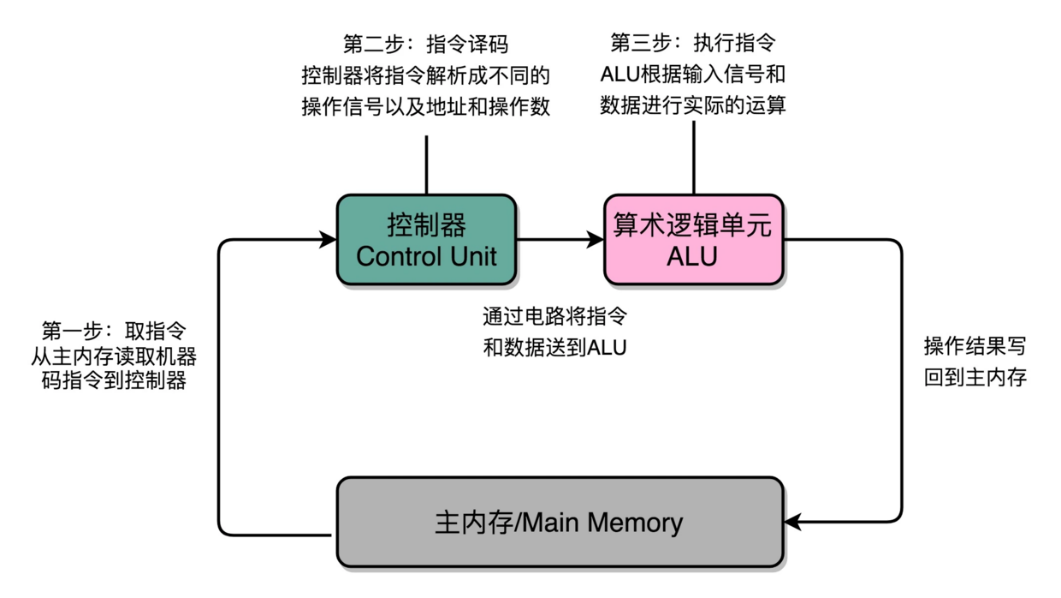
在学习的过程中我使用过Ubuntu和Centos。

* 1. 你对Linux系统有什么认识？

在我心目中，Linux是一种可以和Windows相媲美的操作系统。Linux一切皆文件、完全开源免费、支持多用户和多任务、同时还支持多种架构平台、可靠的安全性、良好的稳定性、具有强大的网络功能、多样图形界面等等都展现其巨大的魅力。通过Linux我更好地了解学习了计算机网络、操作系统等的相关知识，也实践了深度学习、智能机器人技术等应用。

**五、CPU指令执行过程**





1. 首先，我们有一个自动计数器。这个自动计数器会随着时钟主频不断地自增，来作为我们的 PC 寄存器。
2. 在这个自动计数器的后面，我们连上一个译码器。译码器还要同时连着我们通过大量的 D 触发器组成的内存。
3. 自动计数器会随着时钟主频不断自增，从译码器当中，找到对应的计数器所表示的内存地址，然后读取出里面的 CPU 指令。
4. 读取出来的 CPU 指令会通过我们的 CPU 时钟的控制，写入到一个由 D 触发器组成的寄存器，也就是指令寄存器当中。
5. 在指令寄存器后面，我们可以再跟一个译码器。这个译码器不再是用来寻址的了，而是把我们拿到的指令，解析成 opcode 和对应的操作数。
6. 当我们拿到对应的 opcode 和操作数，对应的输出线路就要连接 ALU，开始进行各种算术和逻辑运算。对应的计算结果，则会再写回到 D 触发器组成的寄存器或者内存当中。

模拟代码如下：

1. *// 模拟CPU指令执行过程流水线*
2. #include <iostream>
3. *//取指令（Instruction Fetch，IF）阶段*
4. *//译码（Instruction Decode，ID）阶段*
5. *//执行指令（Execute，EX）阶段*
6. *//访存取数（Memory，MEM）阶段*
7. *//结果写回（Writeback，WB）阶段*
8. *//指令流水线*
9. class InstructionPipeline
10. {
11. public:
12. InstructionPipeline() {}
13. ~InstructionPipeline() {}
14. void IF();
15. void ID();
16. void EX(int a, int b);
17. void MEM();
18. void WB();
19. *//Program Counter Register*
20. int (\*p) (int a, int b);
21. *//Instruction Register*
22. int (\*i) (int a, int b);
23. *//Register*
24. int (\*r) (int a, int b);
25. *//Memory*
26. int m = 0;
27. int t = 0;
28. int c = 0;
29. };
30. *//加法算数指令*
31. int add(int a, int b)
32. {
33. return a + b;
34. }
35. *//IF阶段*
36. void InstructionPipeline::IF()
37. {
38. int (\*t) (int a, int b);*//temp*
39. *//从指令存储器中取指令*
40. t = p;
41. *//下一条指令地址，自动递增；*
42. *//p++;*
43. *//将指令存入指令寄存器*
44. i = p;
45. }
46. *//ID阶段*
47. void InstructionPipeline::ID()
48. {
49. int (\*t) (int a, int b);*//temp*
50. *//从指令寄存器中取指令*
51. t = i;
52. *//译码指令*
53. i = add;
54. *//将译码结果存入译码寄存器*
55. r = i;
56. }
57. *//EX阶段*
58. void InstructionPipeline::EX(int a, int b)
59. {
60. int (\*t) (int a, int b);*//temp*
61. *//从译码寄存器中取译码结果*
62. t = r;
63. *//执行指令*
64. int result = (\*t)(a, b);
65. *//将执行结果存入执行寄存器*
66. m = result;
67. }
68. *//MEM阶段*
69. void InstructionPipeline::MEM()
70. {
71. int temp = 0;
72. *//访存取数*
73. temp = m;
74. *//将访存结果存入访存寄存器*
75. t = temp;
76. }
77. *//WB阶段*
78. void InstructionPipeline::WB()
79. {
80. int temp = 0;
81. *//从访存寄存器中取访存结果*
82. temp = t;
83. *//将结果写回*
84. c = temp;
85. std::cout << "result = " << c << std::endl;
86. }
87. int main()
88. {
89. int a = 1, b = 1;
90. InstructionPipeline CPU;
91. std::cout << "取指..." << std::endl;
92. CPU.IF();
93. std::cout << "译码..." << std::endl;
94. CPU.ID();
95. std::cout << "执行..." << std::endl;
96. CPU.EX(a, b);
97. std::cout << "存结果..." << std::endl;
98. CPU.MEM();
99. CPU.WB();
100. std::cout << "下一条指令地址" << std::endl;
101. return 0;
102. }

结果如下：

