实验报告：pintos安装

课程名称：操作系统实践	年级：2023级	上机实践成绩：
指导教师：张民	姓名：李彤	
上机实践名称：pintos安装	学号：10235101500	上机实践日期：2024.09.16
上机实践编号：	组号：	上机实践时间：

一、目的

安装pintos

二、实验过程

步骤一：安装docker

> 在官网Docker: Accelerated, Containerized Application Development，选择apple chip版本下载

问题1：下载docker安装包

由于理科大楼教室网络较差，所以在使用校园网下载docker安装包时，下载速度特别慢。

解决方法：

一开始，我尝试连接自己的热点来下载docker安装包，但效果并不是很好，下载速度还是很慢。

因为在这节课之前我并没有爬墙的经历，结合本人有限的经验里，我选择连接热点的基础上还连了**学校的VPN** ~~有点走投无路了，开始各种尝试，surprise！~~ ，没想到下载速度变得特别快，原本预计30min的下载时间直接缩减到**30s不到。**

后来我也将这个方法分享给了身边的同学，试了一下发现也行！但是因为没有深究其原因以及广泛实验，我也不太清楚其中的原理，只是觉得自己似乎发现了一个可能有效的下载方式。 ~~也许可以帮助以后修这门课的同学解决类似的问题，但是不保证一定有效。~~

问题2：注册并登录docker

因为自己没有梯子o(╥﹏╥)o，所以一直打不开docker注册的网页。

解决方法：

一位非常热心的同学把他的梯子借我用了，也算是把docker的账号注册好了。

课程结束之后，我也给自己搞了一个梯子。

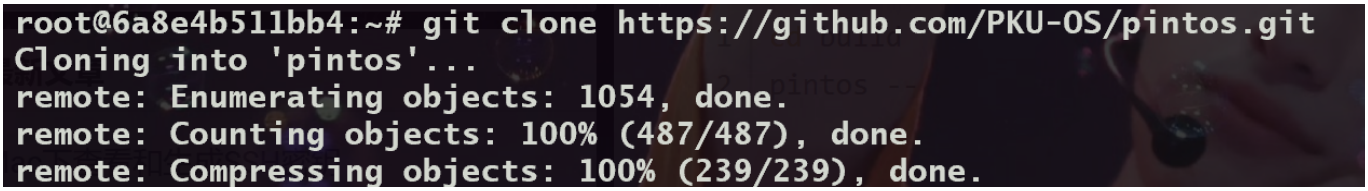
步骤二：拉取镜像

> docker run -it pkuflyingpig/pintos bash 等待一段较长的时间后拉取成功。 成功后会进入一个bash shell。

> - 键入pwd，您会发现您的个人目录在/home/PKUOS下。
> - 键入ls，你会发现有一个包含所有依赖项toolchain目录。

注

因为我上课的时候进度比较慢，所以在拉取镜像之前，已经有同学提出地址的问题，在老师和助教的提醒下，我是直接从github上面复制的http地址，总的来说这一步进行的还算比较顺利。



拉取成功之后，我按照教程的提示先后输入了pwd和ls指令，在得到正确的结果之后我就直接进入了下一步~~（此时还没有好好看教程的内容，为步骤三找不到本机地址埋下了伏笔）~~

步骤三：安装pintos

> 从github上克隆代码

```
git clone git@github.com:PKU-OS/pintos.git
```

> 这个代码是已经配置好toolchain的，可以直接使用不需要自己配置，如果克隆失败，也可以自己配置，参考链接Project Setup。 在此运行docker，注意source要换成pintos的本机地址：

```
docker run -it --rm --name pintos --mount
type=bind,source=absolute/path/to/pintos/on/your/host/machine,target=/home/PKUOS/p
intos pkuflyingpig/pintos bash
```

问题1：找不到pintos本机地址

~~因为前面就是跟着教程做的，所以对于每一步的原因和原理什么也不是很清楚，然后！在*将source换成pintos本机地址*这一步就彻底晕了!找不到本机地址...~~

解决方法：

~~~我先是把C盘和D盘都查找了一遍，发现没有目标文件之后，又重新看了一下这个教程，然后我就在步骤二找到了答案(* ^ ▿ ^ *)
~~~

步骤二：拉取镜像

```
docker run -it pkuflyingpig/pintos bash
```

等待一段较长的时间后拉取成功。

成功后会进入一个bash shell。

- 键入pwd，您会发现**您的个人目录在**/home/PKUOS**下。**
- 键入ls，你会发现**有一个包含所有依赖项**toolchain**目录。**

现在，您在主机计算机中拥有一个微小的Ubuntu操作系统

按照上面的提示，在终端输入pwd，得到了pintos的本机地址！然后把地址复制给source，再次执行指令以后就成功了。~~

~~以上是我最开始的想法，后面发现其实是错误的（简直错得离谱），下面的是更正之后的实验思路~~

问题和分析：

问题： 一开始其实我每太看懂这个教程，我以为步骤二是接着步骤一直接在虚拟机里面完成的，~~实际上不~~是，所以导致我每次docker run之后目录下面都没有pintos文件，于是就要不停地重新clone代码...

分析： 因为docker在每一次运行的时候都会新建一个虚拟机的实例，这就导致即使我在当前的虚拟机里面直接clone了代码，在下一次新创建的实例上也不会有这段代码。这就导致了我不停地clone。

事实上，教程的意思是让我们直接把代码clone到本地

```
24916@psycho MINGW64 ~ (master)
$ git clone git@github.com:PKU-OS/pintos.git
Cloning into 'pintos'...
remote: Enumerating objects: 1054, done.
remote: Counting objects: 100% (735/735), done.
remote: Compressing objects: 100% (393/393), done.
remote: Total 1054 (delta 368), reused 342 (delta 342), pack-reused 319 (from 1)
Receiving objects: 100% (1054/1054), 393.55 KiB | 422.00 KiB/s, done.
Resolving deltas: 100% (452/452), done.
```

然后再执行命令

```
docker run -it --rm --name pintos --mount
type=bind,source=absolute/path/to/pintos/on/your/host/machine,target=/home/PKUOS/p
intos pkuflyingpig/pintos bash
```

将本地文件挂载到虚拟机上，这样就避免了重复clone代码的麻烦

~~以上的分析可能老师在课堂上就已经提到过，但是可能当时我还在搞前面的内容，所以没有听到。~~

在搞清楚这部分内容之后，后面的操作就相对简单很多了，只需要按照教程把所有命令都输一遍就OK了。

```
root@6a8e4b511bb4:~# ls
pintos  toolchain
root@6a8e4b511bb4:~# cd pintos/src/threads/
root@6a8e4b511bb4:~/pintos/src/threads# make
mkdir -p build/devices
mkdir -p build/lib
mkdir -p build/lib/kernel
mkdir -p build/lib/user
mkdir -p build/tests/threads
```

```
make[1]: Leaving directory '/home/PKUOS/pintos/src/threads/build'
root@6a8e4b511bb4:~/pintos/src/threads# cd build
root@6a8e4b511bb4:~/pintos/src/threads/build# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/vaJ_g2tWlf.dsk -m
graphic -monitor null
Pintos hda1
Loading...........
Kernel command line:
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  130,867,200 loops/s.
Boot complete.
```

```
root@a25eeeff855a:~/pintos/src/threads/build# pintos -- -q run alarm-multiple
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/Xa4tWgv5kD.dsk -m 4 -net none -nographic -monitor null
Pintos hda1
Loading...........
Kernel command line: -q run alarm-multiple
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  157,081,600 loops/s.
Boot complete.
Executing 'alarm-multiple':
(alarm-multiple) begin
(alarm-multiple) Creating 5 threads to sleep 7 times each.
(alarm-multiple) Thread 0 sleeps 10 ticks each time,
(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.
(alarm-multiple) If successful, product of iteration count and
(alarm-multiple) sleep duration will appear in nondescending order.
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 0: duration=10, iteration=5, product=50
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 0: duration=10, iteration=6, product=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 577 ticks
Thread: 0 idle ticks, 577 kernel ticks, 0 user ticks
Console: 2954 characters output
Keyboard: 0 keys pressed
Powering off...
root@a25eeeff855a:~/pintos/src/threads/build# _
```

心得体会

在这个实验中，我收获的最大的教训就是要好好看教程，一定要弄懂每一个步骤都是在什么环境下完成的，其中的原因和原理都是什么，否则的话就会像这次一样兜很多圈子。

但是！自己一步一步去摸索、尝试，到最后弄懂、弄好一个实验，还是很有成就感的。如果我一开始就是按照正确的教程走的，虽然实验过程变得简单顺利了，但我可能也就不会有更深入的思考和学习，不会去深究每一个步骤背后的内涵究竟是什么，也就学不到更多的东西了。

除此之外，通过这个实验我对docker这一个平台的工作原理有了一个初步的了解：

- 如果把image理解成可执行程序，那么container就是运行这个程序的实例，dockerfile就是用来构建image的源代码，而docker就是“编译器”。
- 每次执行docker run指令之后都会创建一个全新的container实例，这些实例都具有一模一样的环境变量，并且彼此之间是相互独立的。这样程序运行时就不会因为环境不一样出错，报错后也不会牵连其他容器中的程序。