

实验报告：Pintos 安装

课程名称：操作系统	年级：2023 级本科	上机实践成绩：
指导教师：张民	姓名：张梓卫	
上机实践名称：Pintos 安装	学号：10235101526	上机实践日期：
上机实践编号：（1）	组号：	上机实践时间：

目录	
一 实验目的	1
二 内容与设计思想	1
三 使用环境	2
四 实验过程与分析	2
1 使用 Docker 拉取镜像	2
2 从 Github 克隆代码	2
3 使用 cd、pwd、ls 命令等进入 threads	3
3.1 cd 命令	3
3.2 ls 命令	3
3.3 pwd 命令	3
4 进入 threads 目录	3
5 编译 Pintos	3
6 运行 Pintos	4
7 完成 alarm-multiple	4
7.1 进入 build 文件夹	4
7.2 Run alarm-multiple	4
8 实验结束，进行后续预热	5
五 实验结果总结	6
1 自定义 Windows 命令实现 Alias	6
六 附录	7

一 实验目的

本次实验为操作系统的第一次实验，目的是安装并运行 Pintos 操作系统，并熟悉操作系统的基本概念、原理和实践方法。实验中，还对当下流行的 Docker 容器技术进行了实践，基于 PKU-OS 进行教学，让我们能够顺便熟悉 Linux 操作系统中的部分命令，为以后的开发打下基础。

二 内容与设计思想

利用 Docker 在 Windows 11 上安装基于 x86 的 Pintos（斯坦福大学）。

三 使用环境

使用 Docker v27.1.1 进行 Pintos 的安装实验，基于 Windows 11 操作系统使用 WSL2。
实验报告使用 L^AT_EX 进行撰写，使用 Vim 编辑器进行文本编辑。

四 实验过程与分析

1 使用 Docker 拉取镜像

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

拉取镜像

```
C:\Users\26421>docker run -it pkuflyingpig/pintos bash
root@ce8194b42c3e:~#
```

图 1: 拉取镜像后进入 Docker 容器

2 从 Github 克隆代码

注意，文档里写的这个无法拉取，需要使用 SSH 协议，需要在 GitHub 上设置 SSH 密钥。

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

克隆代码

所以使用了助教发送的新地址：

```
1 > git clone https://gitee.com/duerwuyi/pintos.git
```

克隆代码

Clone 代码结果如下：

```
root@ce8194b42c3e:~# git clone https://gitee.com/duerwuyi/pintos.git
Cloning into 'pintos'...
remote: Enumerating objects: 1054, done.
remote: Total 1054 (delta 0), reused 0 (delta 0), pack-reused 1054
Receiving objects: 100% (1054/1054), 393.93 KiB | 6.25 MiB/s, done.
Resolving deltas: 100% (450/450), done.
```

图 2: 克隆代码

其中，Docker 中的 Log 显示了克隆的进度

```
2024-09-26 23:39:38 root@ce8194b42c3e:~# git clone https://gitee.com/duerwuyi/pintos.git
2024-09-26 23:39:38 Cloning into 'pintos'...
2024-09-26 23:39:39 remote: Enumerating objects: 1054, done.
2024-09-26 23:39:39 Receiving objects: 0% (1/1054)
Receiving objects: 1% (11/1054)
Receiving objects: 2% (22/1054)
Receiving objects: 3% (32/1054)
Receiving objects: 4% (43/1054)
Receiving objects: 5% (53/1054)
Receiving objects: 6% (64/1054)
Receiving objects: 7% (74/1054)
Receiving objects: 8% (85/1054)
Receiving objects: 9% (95/1054)
Receiving objects: 10% (106/1054)
```

图 3: 克隆进度

3 使用 cd、pwd、ls 命令等进入 threads

首先，对 cd、pwd、ls 等命令进行复习：

3.1 cd 命令

- cd 进入用户主目录；
- cd ~ 进入用户主目录；
- cd - 返回进入此目录之前所在的目录；
- cd .. 返回上级目录（若当前目录为 “/”，则执行完后还在 “/”；“..” 为上级目录的意思）；
- cd ../.. 返回上两级目录；
- cd !\$ 把上个命令的参数作为 cd 参数使用。

3.2 ls 命令

- -a: 列出目录下的所有文件，包括以 . 开头的隐藏文件；
- -l: 除了文件名之外，还将文件的权限、所有者、文件大小等信息详细列出来。

3.3 pwd 命令

全称 Print Working Directory，显示当前工作目录的完整路径。

4 进入 threads 目录

```
1 > cd pintos
2 > ls
3 > cd src
4 > ls
5 > cd threads
```

进入 threads 目录

```
root@ce8194b42c3e:~# pwd
/home/PKUOS
root@ce8194b42c3e:~# ls
pintos toolchain
root@ce8194b42c3e:~# cd pintos
root@ce8194b42c3e:~/pintos# ls
README.md docs src
root@ce8194b42c3e:~/pintos# cd src
root@ce8194b42c3e:~/pintos/src# ls
LICENSE      Makefile      Makefile.kernel  devices  filesys  misc  threads  utils
Make.config  Makefile.build  Makefile.userprog  examples  lib      tests  userprog  vm
root@ce8194b42c3e:~/pintos/src# cd threads
root@ce8194b42c3e:~/pintos/src/threads#
```

图 4: 进入 threads 目录

这部分疯狂使用 ls 命令是为了在命令行中没有界面时能够及时获取当前文件夹的信息。

5 编译 Pintos

在当前目录使用 make 指令，即可编译 Pintos。

```
1 > make
```

编译 Pintos

```

C:\Windows\system32\cmd.exe
ck-protector -nostdinc -I../.. -I../lib -I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers -MM -MF tests/threads/mlfqs-recent-ld
i386-elf-gcc -c ../tests/threads/mlfqs-fair.c -o tests/threads/mlfqs-fair.o -m32 -g -msoft-float -O0 -fno-stack-protector -nostdinc -I../.. -I../lib -I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers -MM -MF tests/threads/mlfqs-fair.d
i386-elf-gcc -c ../tests/threads/mlfqs-block.c -o tests/threads/mlfqs-block.o -m32 -g -msoft-float -O0 -fno-stack-protector -nostdinc -I../.. -I../lib -I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers -MM -MF tests/threads/mlfqs-block.d
i386-elf-ld -melf_i386 -T threads/kernel.lds.s -o kernel.o threads/start.o threads/init.o threads/thread.o threads/switch.o threads/interrupt.o threads/intr-stubs.o threads/synch.o threads/palloc.o threads/malloc.o devices/pit.o devices/timer.o devices/kbd.o devices/vga.o devices/serial.o devices/block.o devices/partition.o devices/ide.o devices/input.o devices/intc.o devices/rtc.o devices/shutdown.o devices/speaker.o lib/debug.o lib/random.o lib/stdio.o lib/stdlib.o lib/string.o lib/arithmetic.o lib/ustar.o lib/kernel/debug.o lib/kernel/list.o lib/kernel/bitmap.o lib/kernel/hash.o lib/kernel/console.o tests/threads/tests.o tests/threads/alarm-wait.o tests/threads/alarm-simultaneous.o tests/threads/alarm-priority.o tests/threads/alarm-zero.o tests/threads/alarm-negative.o tests/threads/priority-change.o tests/threads/priority-donate-one.o tests/threads/priority-donate-multiple.o tests/threads/priority-donate-multiple2.o tests/threads/priority-donate-nest.o tests/threads/priority-donate-sema.o tests/threads/priority-donate-lower.o tests/threads/priority-fifo.o tests/threads/priority-preempt.o tests/threads/priority-sema.o tests/threads/priority-condvar.o tests/threads/priority-donate-chain.o tests/threads/mlfqs-load-1.o tests/threads/mlfqs-load-60.o tests/threads/mlfqs-load-avg.o tests/threads/mlfqs-recent-1.o tests/threads/mlfqs-fair.o tests/threads/mlfqs-block.o
i386-elf-objdump -S kernel.o > kernel.asm
i386-elf-nm -n kernel.o > kernel.sym
i386-elf-objcopy -R .note -R .comment -S kernel.o kernel.bin
i386-elf-gcc -c ../threads/loader.S -o threads/loader.o -Wa,--gstabs,--32 -nostdinc -I../.. -I../lib
i386-elf-ld -melf_i386 -N -e 0 -Ttext 0x7c00 -o loader.out threads/loader.o
i386-elf-objdump -S loader.out > loader.asm
i386-elf-objcopy -S -O binary -j .text loader.out loader.bin
rm loader.out
make[1]: Leaving directory '/home/PKUOS/pintos/src/threads/build'
root@ce8194b42c3e:~/pintos/src/threads#

```

图 5: 编译 Pintos

通过观察可以发现，输出结果中提示，此时已经离开了 build 目录。

6 运行 Pintos

```
> pintos —
```

运行 Pintos

输入该命令，输出的提示信息如下，代表 Boot 已完成，Pintos 已成功编译。

```

root@ce8194b42c3e:~/pintos/src/threads# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/smPLun9upD.dsk -m 4 -net none -no-
graphic -monitor null
Pintos hdai
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... 209,510,400 loops/s.
Boot complete.

```

图 6: 运行 Pintos

7 完成 alarm-multiple

7.1 进入 build 文件夹

当完全 Boot 完成后，会发现终端卡在了 Boot Complete，而没有退出回到可以交互的状态。

此时，使用 Ctrl + C 组合键打断此时的状态，然后回退到可交互的 Bash 终端中，可以发现，回退到了 /pintos/src/thread 通过 ls 指令查看当前文件夹，并且通过 cd build 进入 build 文件夹内

```

Boot complete.
qemu-system-i386: terminating on signal 2

root@ce8194b42c3e:~/pintos/src/threads# ls
Makevars  flags.h  interrupt.c  intr-stubs.h  loader.S  malloc.h  pte.h  switch.h  thread.c
Makefile  init.c  interrupt.h  io.h  loader.h  palloc.c  start.S  synch.c  thread.h
build     intr-stubs.S  kernel.lds.S  malloc.c  palloc.h  switch.S  synch.h  vaddr.h
root@ce8194b42c3e:~/pintos/src/threads# cd build
root@ce8194b42c3e:~/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/rjECxNwxb0.dsk -m 4 -net none -no-
graphic -monitor null
Pintos hdai
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... 209,510,400 loops/s.
Boot complete.

```

图 7: 进入 build 文件夹

7.2 Run alarm-multiple

```
1 > pintos — -q run alarm-multiple
```

Run alarm-multiple

```
C:\Windows\system32\cmd.e. X + v
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 209,510,400 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 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 2: duration=30, iteration=1, 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 0: duration=10, iteration=6, product=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 1: duration=20, iteration=4, 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 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 3: duration=40, iteration=3, 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 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 4: duration=50, iteration=4, 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: 575 ticks
Thread: 0 idle ticks, 575 kernel ticks, 0 user ticks
Console: 2954 characters output
Keyboard: 0 keys pressed
Powering off...
root@ce8194b42c3e:~/pintos/src/threads/build#
```

图 8: 运行 alarm-multiple

8 实验结束, 进行后续预热

在 Docker 中的 Container 里运行时, 交互的命令不是 Windows, 使用的是 Exit 命令来退出当前的 Container。

• 退出交互模式:

- 使用 'exit' 命令: 退出当前的交互终端并关闭容器 (如果容器是通过交互模式启动的)。
- 使用快捷键 'Ctrl + D': 在没有挂起的进程时, 按下 'Ctrl + D' 也会退出交互模式。

• 保持容器运行并退出交互模式:

- 使用快捷键 'Ctrl + P' 然后 'Ctrl + Q': 容器继续在后台运行, 你可以从交互模式中分离。
通过以下命令重新连接到容器:

```
docker attach <container_id_or_name>
```

- 从宿主系统停止容器:

- 列出所有正在运行的容器: `docker ps`
- 使用 ‘`docker stop`’ 命令停止容器:

```
docker stop <container_id_or_name>
```

- 在容器内部运行命令关闭:

- 使用 ‘`shutdown`’ 命令关闭容器内的操作系统: `shutdown now`
- 使用 ‘`halt`’ 命令停止当前容器系统:

五 实验结果总结

通过在虚拟机中运行的方式, 无法将本地文件与 Container 内的文件相关联, 于是我选择在主机上 Clone 代码, 然后挂载到 Docker 中运行。

首先, Clone 代码到本地 (Path: C:

Users
26421
pintos)

使用如下代码将 Pintos 目录挂载到 Docker 中, 以后所有的变化都会同步到本地。

```
1 docker run -it --rm --name pintos --mount type=bind,source=C:\Users\26421\pintos,target=/home/PKUOS/pintos
pkufllyingpig/pintos bash
```

这一部分是参考官方文档: <https://pkufllyingpig.gitbook.io/pintos/getting-started/environment-setup>

因为我们所有的文件更改都会同步到 Windows 系统中, 所以要加入 `-rm` 参数告诉 Docker 每次运行结束后就删除 Container, 不然会占用很多的存储空间。

Now when you `ls`, you will find there is a new directory called `pintos` under your home directory in the container, **it is shared by the container and your host computer i.e. any changes in one will be synchronized to the other instantaneously.**

现在当你 `ls` 时, 你会发现在容器的主目录下有一个名为 `pintos` 的新目录, 它被容器和你的主机共享, 即其中一个的任何更改都会立即同步到另一个。

图 9: 挂载目录

1 自定义 Windows 命令实现 Alias

由于每一次我们都需要在 Windows Bash 终端中输入同样的命令以挂载到 Docker 中, 所以我们可以自定义 Windows 命令, 以省略每一次输入大量命令的步骤。

新建批处理文件

新建 PintosPintos.txt 于 Windows Desktop 中, 输入以下命令:

```
1 @echo off
2 docker run -it --rm --name pintos --mount type=bind,source=C:\Users\26421\pintos,target=/home/PKUOS/pintos
pkufllyingpig/pintos bash
```

PintosPintos.txt

其中, @echo off 是用来隐藏批处理文件执行时的命令行输出, 只显示实际的执行结果, 使脚本运行时的输出更加简洁明了。

之后, 将这个文本文件重命名为 Pintos.bat, 当桌面目录添加至系统的环境变量 PATH 中, 于是使用 Win + R, 输入 cmd, 再输入 Pintos.bat 即可运行 Pintos 容器。

六 附录

本实验参考了以下资源:

- PKU-OS 文档: <https://pkufllyingpig.gitbook.io/pintos/getting-started/environment-setup>
- Docker 官方文档: <https://docs.docker.com/>