Pintos实验记录

安装

电脑环境

Ubuntu20.04.1 LTS

Linux version 5.4.0-42-generic

检查安装环境

```
#GCC
gcc --version
#GNU binutils
addr2line --version
ar --version
ld --version
objcopy --version
ranlib --version
#Perl
perl --version
#GNU make
make --version
#QEMU
qemu-system-aarch64 --version
#GDB
gdb --version
#X
sudo X --version
```

若不存在qemu则按如下命令安装或者选择不安装qemu而用bochs

```
sudo apt-get install qemu
```

安装并测试PintOS

下载pintos源码并解压

```
#以下为旧版本  安装会有一些问题
#wget http://www.stanford.edu/class/cs140/projects/pintos/pintos.tar.gz
#tar -xzvf pintos.tar.gz

#安装新版本
git clone git://pintos-os.org/pintos-anon

#清除版本控制信息
cd pintos-anon
find . -name ".git" | xargs rm -Rf
```

编辑环境变量

```
gedit ~/.bashrc
```

插入 `export  PATH=$PATH:/usr/bin:/home/.../pintos-anon/src/utils`（如果是 `zsh` 则配置 `.zashrc`）

更新配置

```
source ~/.bashrc
```

编译utils

1. 修改 `/utils/pintos-gdb` 中的 `GDBMACROS` 为当前路径 `GDBMACROS=/home/.../pintos-anon/src/misc/gdb-macros`
2. 修改 `/utils/Makefile` 中的 `LOADLIBES` 为 `LDLIBS`
3. 在 `/src/utils` 中执行make,编译成功则执行下一步，若编译失败则查看存在问题中的解决方案。

编译threads

1. 修改 `/src/threads/MAke.vars`，更改 `bochs` 为 `qemu`
2. 在 `/src/threads` 下执行 `make`

编辑pintos

`/utils/pintos`

- 103行：更改 `bochs` 为 `qemu`
- 259行：更改 `kernel.bin` 为 `/home/.../src/threads/build/kernel.bin`
- 623行：更改 `qemu-system-i386` 为 `qemu-system-x86_64`

编辑pintos.pm

`/utils/Pintos.pm`

- 362行：更改 `loader.bin` 为 `/home/.../src/threads/build/loader.bin`

运行pintos

`pintos run alarm-multiple`

存在问题

如果在`/src/utils`中执行make报错

```
squish-pty.c:10:10: fatal error: stropts.h: No such file or directory
```

这是因为LInux版本较高，老版本的Linux会包含这个文件，但是有些新的Linux版本已经不再包含了。解决方案可以在`/usr/include/`下建立一个空的`stropts.h`文件，或者注释掉代码中与之相关的部分（不要从老版本的Linux中直接拷贝过来，拷贝过来的stropts.h文件中，可能还会包含别的找不到的头文件）。

如果引入stropts.h后仍然再次报错，则需要根据报错注释掉一些代码，仍然是版本问题。需要注释掉`squish-pty.c`中掉第288到293行

参考链接

https://web.stanford.edu/class/cs140/projects/pintos/pintos_12.html#SEC166

https://github.com/kumardeepakr3/PINTOS-Ubuntu

Thread

Test的运行过程

在命令行键入`pintos run alarm-single(test name)`时：

从`init.c main`开始 先是一堆初始化，然后读取命令行参数，运行函数中134行`run_actions(argv)`;跳转至对应函数做参数解析，如果是`test`则运行当前283行的`run_task`并跳转`run_test`函数其匹配`tests.c`中的各项测试，并运行匹配到的测试名即`tests`中的字典中的项。

编码过程

Alarm Clock

`threads`先测试，发现能过7个。线程捐赠出了问题.但是其实前面的也是忙等待，一直在`thread_yeild()`。先解决忙等待的问题，增加`thread_sleep`状态，为线程增加`sleep_ticks`对象，在中断时`--`，`=0`则进入就绪状态。这就解决忙等待。然后是优先级捐赠。

Priority Scheduling

跑几个测试看看，以一个为例，应该是优先级33，但实际31，调试`thread_create`函数发现没问题，即`thread_get_priority`函数的逻辑也没问题。也就是说代码的底层逻辑都是对的，那问题出在锁上面。 `acquire0`优先级为31 `acquire1`为32 由于`acquire0`加锁了，则`acquire2`阻塞，因此需要：当发现高优先级的任务因为低优先级任务占用资源而阻塞时，就将低优先级任务的优先级提升到等待它所占有的资源的最高优先级任务的优先级。

查看代码，发现问题出在`thread_create`中进程创建时，进程创建时，最后执行了一个`thread_unblock`函数，其调用`list_push_back`将进程放入就绪队列中。但观察该函数发现其并没有按照优先级插入进程，而是直接插入，这显然需要改。改的思路有两个：

- 按优先级插入队列
- 拿出的时候按照优先级取出。

由于系统函数已经存在`list_insert_ordered`函数，我们需要采用该函数实现插入，并补全该函数。即按优先级插入队列。

完成按照优先级插入队列的要求后，因为存在抢占，因此，每次对优先级的改变都要重新调度一下进程，选择优先级最高的进程运行。但如果低优先级进程持有锁，高优先级进程请求锁，那就不能简单的调度，需要提升低优先级进程的优先级，提升到它持有的最高级别的锁的优先级，同时在释放锁后降低到原来的优先级。同时，因为是优先级调度，那信号量的PV也需要优先级，不然唤醒的就不是优先级最高的线程。

Advanced Scheduler

多级反馈调度跟着实验书公式来差不多

$$priority = PRI\_MAX - (recent\_cpu/4) - (nice * 2)$$

$$recent\_cpu = (2 * load\_avg)/(2 * load\_avg + 1) * recent\_cpu + nice$$

$$load\_avg = (59/60) * load\_avg + (1/60) * ready\_threads$$

因为`pintos`不支持浮点运算，那直接用`int`型好了，定义个`nice`，定义个`recent_cpu`。用`nice`和轮转值结合上面公式来进行调度，当前`recent_cpu`迭代次数越多，优先级越低。

参考链接

https://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html#SEC15

https://www.cnblogs.com/laiy/p/pintos_project1_thread.html

https://blog.csdn.net/xwyzsn/article/details/107396398