南 京 理 工 大 学

智能科学与技术

软件课程设计(III)

姓　　名: 高　宏　艳

学　　号: 919106840202

指导老师: 杜　鹏　桢

目录

1 设计要求

本次实验基于 padavan-ng 项目（项目地址：https://gitlab.com/padavan-ng/padavan-ng），主要有以下几点要求：

1 编译系统，下载并提交，验收的时候，我会找 MIPS 板子，让大家下载验证
2 编译一个内核模块，内容任选，随系统提交，要求内容可体现在系统日志中
3 编译一个应用模块，内容任选，以固件形式随系统提交，要求可在自指定文件中有结果显示
4 trunk/linux-3.4.x/kernel/sched/core.c 是进程相关的一个核心文件，同学们在必要函数前加注释，画出大体调度流程；trunk/linux-3.4.x/mm/page_alloc.c 是内存管理相关的一个核心文件，同学们在必要函数前加注释，画出大体分配流程

2 项目环境

2.1 硬件和软件环境

硬件环境	软件环境
● **硬盘 20G** ● **内存 2G**	● Ubuntu18.04 Server ● VMware Workstation 16 Player ● XShell7

2.2 项目部署

2.2.1 安装虚拟机以及 Ubuntu 系统

虚拟机版本没有太多硬性要求，老师文档给的虚拟机版本相对较低（提取地址：https://pan.baidu.com/s/1VCw2_1XSE7_Oh_L9hwF0ow），考虑到系统兼容性，本次实验中使用的虚拟机是 VMware Workstation 16 Player.

安装 Ubuntu 使用的镜像文件来自科大镜像网站，虽然版本不同但是安装步骤同老师给的文档，这里不多赘述。

2.2.2 系统配置

执行

sudo passwd root

第一次输入 ado 密码，用于验证 ado 用户

第二次输入 root 密码
第三次输入 root 密码 验证

显示成功后，

su - root

用编辑器(vi 或 nano)打开/etc/ssh/sshd_config

```
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Port 22
#AddressFamily any
ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
PermitRootLogin yes
StrictModes no
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile     .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
                                                              32,19         9%
```

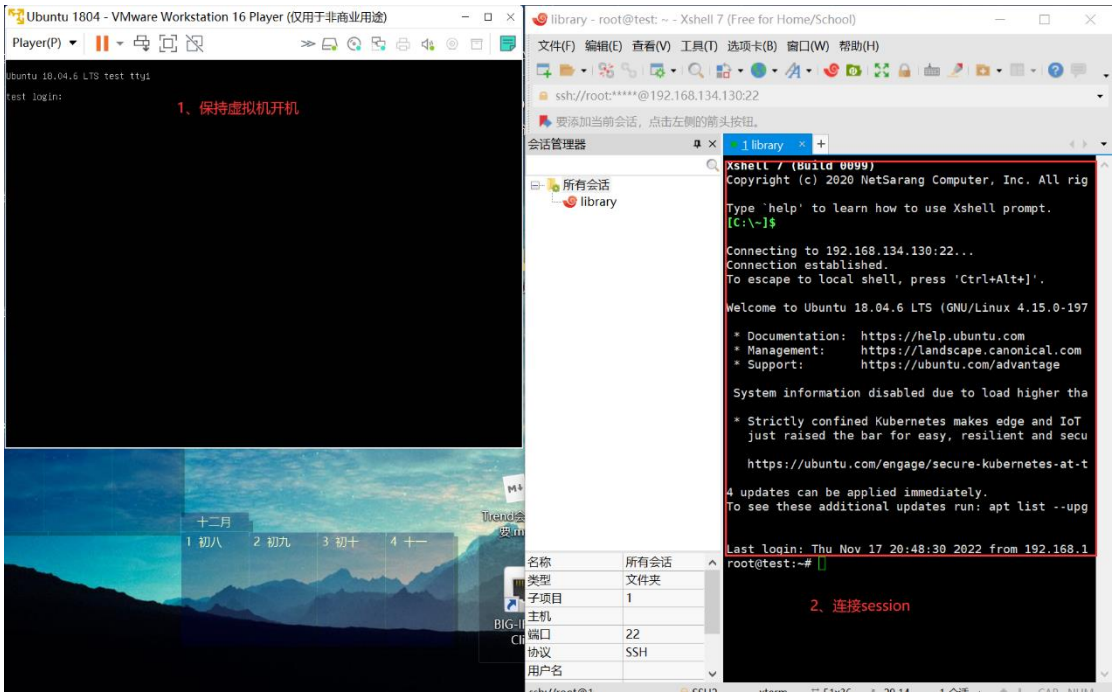
彩色部分，按照上面修改。
然后输入：

systemctl restart ssh
exit *#退出 root*
exit *#退出 ado*

最小化虚拟机，下面要在保证虚拟机开启的过程中实现终端 XShell7 和虚拟机的交互。

2.2.3 终端配置

按照老师给的教程配置即可，这边没有出现问题，连接成功后界面如下：



2.2.4 实验配置

1、vim 设置

将下面的命令逐行从 XShell7 输入

```
echo "" >> /etc/vim/vimrc
echo "" >> /etc/vim/vimrc
echo "set nocp" >> /etc/vim/vimrc
echo "set ru" >> /etc/vim/vimrc
echo "set background=light" >> /etc/vim/vimrc
echo "set nobk" >> /etc/vim/vimrc
echo "set is" >> /etc/vim/vimrc
echo "syn on" >> /etc/vim/vimrc
echo "set backspace=indent,eol,start" >> /etc/vim/vimrc
echo "set whichwrap=b,s,<,>,[,]" >> /etc/vim/vimrc
echo "set sw=4" >> /etc/vim/vimrc
echo "set ts=4" >> /etc/vim/vimrc
echo "set lbr" >> /etc/vim/vimrc
echo "set sm" >> /etc/vim/vimrc
echo "set cin" >> /etc/vim/vimrc
echo "set softtabstop=4" >> /etc/vim/vimrc
echo "set autoindent" >> /etc/vim/vimrc
echo "set number" >> /etc/vim/vimrc
echo "set hls" >> /etc/vim/vimrc
echo "set fo+=mB" >> /etc/vim/vimrc
echo "set vb t_vb=" >> /etc/vim/vimrc
echo "set nobackup" >> /etc/vim/vimrc
echo "" >> /etc/vim/vimrc
```

2、ssh 设置

逐行输入下面两条命令

```
systemctl reenable ssh
systemctl restart ssh
```

运行结果如下：

```
root@test:~# systemctl reenable ssh
Removed /etc/systemd/system/multi-user.target.wants/ssh.ser
ce.
Removed /etc/systemd/system/sshd.service.
Created symlink /etc/systemd/system/sshd.service → /lib/sys
md/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants
sh.service → /lib/systemd/system/ssh.service.
root@test:~# systemctl restart ssh
```

3、profile 设置

将下面的命令逐行输入至命令行

```
echo "alias chkconfig=\"systemctl list-units --type=service\"" >> /etc/profile
echo "alias astylec=\"astyle -A1t4xkxVSNwYM80pDHk1W1yxLnvz2\"" >> /etc/profile
echo "alias tailf=\"tail -f\"" >> /etc/profile
echo "ulimit -SHn 1000000" >> /etc/profile
echo "ulimit -SHu   65535" >> /etc/profile

echo "fs.file-max = 1000000" >> /etc/sysctl.conf

echo "* soft nofile   999999" >> /etc/security/limits.conf
echo "* hard nofile  1000000" >> /etc/security/limits.conf
echo "* soft nproc     65535" >> /etc/security/limits.conf
echo "* hard nproc     65535" >> /etc/security/limits.conf
echo "net.ipv4.ip_forward=1"  >> /etc/sysctl.conf
```

运行结果如下：

```
Removed /etc/systemd/system/multi-user.target.wants/ssh.servi
ce.
Removed /etc/systemd/system/sshd.service.
Created symlink /etc/systemd/system/sshd.service → /lib/syste
md/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants/s
sh.service → /lib/systemd/system/ssh.service.
root@test:~# systemctl restart ssh
root@test:~# echo "alias chkconfig=\"systemctl list-units --t
ype=service\"" >> /etc/profile
root@test:~# echo "alias astylec=\"astyle -A1t4xkxVSNwYM80pDH
k1W1yxLnvz2\"" >> /etc/profile
root@test:~# echo "alias tailf=\"tail -f\"" >> /etc/profile
root@test:~# echo "ulimit -SHn 1000000" >> /etc/profile
root@test:~# echo "ulimit -SHu   65535" >> /etc/profile
root@test:~# echo "fs.file-max = 1000000" >> /etc/sysctl.conf
root@test:~# echo "* soft nofile   999999" >> /etc/security/l
imits.conf
root@test:~# echo "* hard nofile  1000000" >> /etc/security/l
imits.conf
root@test:~# echo "* soft nproc     65535" >> /etc/security/l
imits.conf
root@test:~# echo "* hard nproc     65535" >> /etc/security/l
imits.conf
root@test:~# echo "net.ipv4.ip_forward=1"  >> /etc/sysctl.con
f
root@test:~#
```

4、设置本地时间

逐行运行下面的命令

```
rm /etc/localtime
ln -s /usr/share/zoneinfo/PRC /etc/localtime
```

5、zh-CN 设置

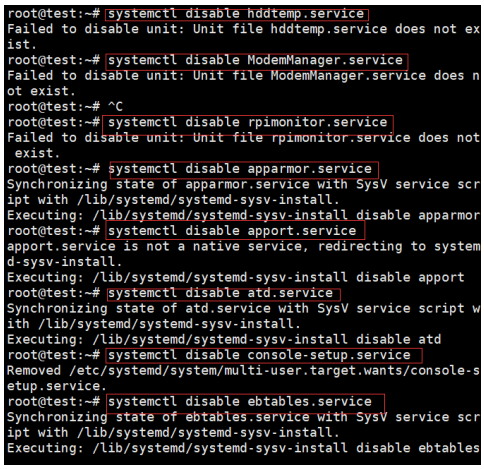
执行 *dpkg-reconfigure locales*，选中 zh_CN.UTF-8 UTF-8,回车后继续选择 zh_CN.UTF-8,然后在命令行输入

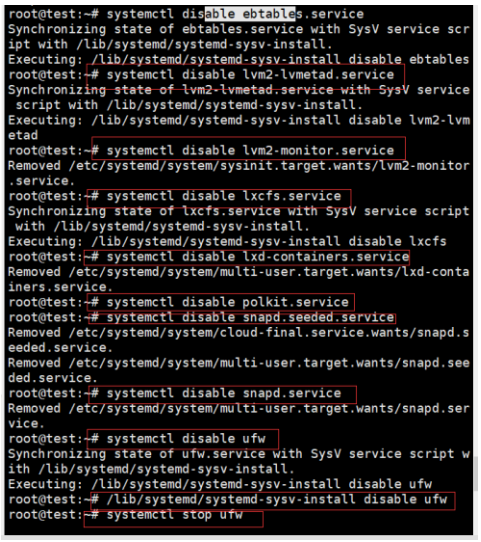
```
echo "LANGUAGE=zh_CN:zh" >> /etc/default/locale
echo "LC_ALL=zh_CN.UTF-8" >> /etc/default/locale
```

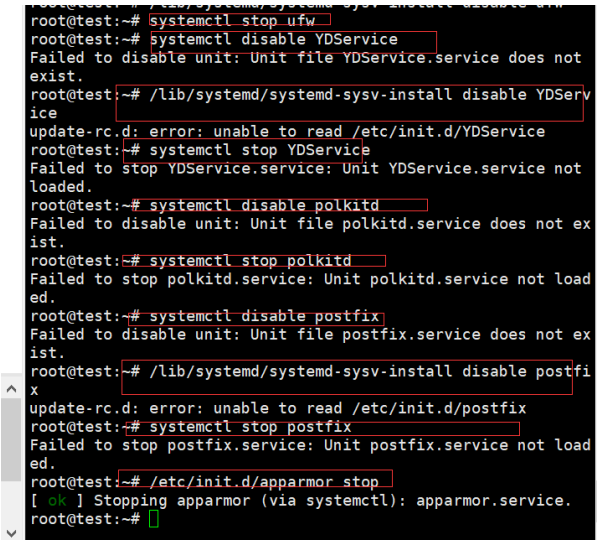
6、apt 设置

命令行执行 *apt-get update && apt-get upgrade -y && apt-get dist-upgrade -y && apt-get autoremove -y && apt-get clean*

7、禁用一些不必要的服务







8、删除无用目录

```
rm /lost+found /media -rf
```

9、安装实验所需要的包

```
apt-get install language-pack-zh-hans apt-utils gcc g++ gdb binutils build-essential make bison flex autoconf automake patch gawk bzip2 gettext pkg-config libtool zlib1g-dev libgmp3-dev libmpfr-dev libmpc-dev libreadline-dev vsftpd lrzsz lftp telnet curl astyle exfat-fuse ethtool sysstat bridge-utils p7zip-full
apt-get install git gperf texinfo python-docutils mc autopoint
rm /lost+found /media -rf
```

10、清理重启

```
apt-get clean && reboot
```

3 项目实现

3.1 编译系统

1、首先需要创建目录 test1,利用 git 将项目克隆到本机，输入命令如下：

```
cd /root
mkdir test1
cd test1
git clone https://gitlab.com/padavan-ng/padavan-ng.git
```

```
Last login: Wed Oct 19 15:47:53 2022 from 192.168.134.1
root@test:~# cd test1
root@test:~/test1# git clone https://gitlab.com/padavan-ng/padavan-ng.git
正克隆到 'padavan-ng'...
remote: Enumerating objects: 119768, done.
remote: Total 119768 (delta 0), reused 0 (delta 0), pack-reused 119768
接收对象中: 100% (119768/119768), 352.30 MiB | 7.74 MiB/s, 完成.
处理 delta 中: 100% (31642/31642), 完成.
正在检出文件: 100% (103391/103391), 完成.
root@test:~/test1#
```

2、接着编译工具链

在本地实验花费时间 36min26s,编译成功截图如下:

```
[INFO ]  Installing pass-2 core C gcc compiler: done in 584.85s (at 22:00)
[INFO ]  ============================================================
[INFO ]  Installing C library
[INFO ]    ============================================================
[INFO ]    Building for multilib 1/1: ''
[EXTRA]      Copying sources to build dir
[EXTRA]      Applying configuration
[EXTRA]      Cleaning up startfiles
[EXTRA]      Building C library
[EXTRA]      Installing C library
[INFO ]    Building for multilib 1/1: '': done in 126.66s (at 24:07)
[INFO ]  Installing C library: done in 126.75s (at 24:07)
[INFO ]  ============================================================
[INFO ]  Installing final gcc compiler
[EXTRA]    Configuring final gcc compiler
[EXTRA]    Building final gcc compiler
[EXTRA]    Installing final gcc compiler
[EXTRA]    Housekeeping for final gcc compiler
[EXTRA]      '' --> lib (gcc)   lib (os)
[INFO ]  Installing final gcc compiler: done in 735.11s (at 36:22)
[INFO ]  ============================================================
[INFO ]  Checking dynamic linker symlinks
[EXTRA]    Checking dynamic linker for multilib ''
[INFO ]  Checking dynamic linker symlinks: done in 0.23s (at 36:22)
[INFO ]  ============================================================
[INFO ]  Finalizing the toolchain's directory
[INFO ]    Stripping all toolchain executables
[EXTRA]    Installing the populate helper
[EXTRA]    Installing a cross-ldd helper
[EXTRA]    Creating toolchain aliases
[EXTRA]    Removing installed documentation
[INFO ]  Finalizing the toolchain's directory: done in 3.76s (at 36:26)
[INFO ]  Build completed at 20221020.090400
[INFO ]  (elapsed: 36:25.81)
[INFO ]  Finishing installation (may take a few seconds)...
[36:26] / ====================All IS DONE!=======================
root@test:~/test1/padavan-ng/toolchain#
```

3、编译系统

返回上层 trunk 目录，执行如下代码:

```
cp configs/templates/youku/yk_l1_full.config .config
./build_fireware.sh
```

编译好的系统在 trunk/images/YK-L1_3.4.3.9L-100.trx，可以用 *sz* 命令将生成的.trx 文件发送到电脑本地进行保存

3.2 编译一个内核模块

1、首先进入项目目录中的`/trunk/linux-3.4.x/drivers`目录下，新建一个自己的内核工程文件

```
cd ./trunk/linux-3.4.x/drivers    # 进入drivers目录
mkdir ghykernel                   # 新建自己的工程目录，这里我起名为ghykernel
```

2、接着，进入该目录，新建三个工程文件 xx.c，Kconfig，Makefile

```
cd ghykernel
touch ghykernel.c Kconfig Makefile # 第一个.c文件名称不限，Kconfig和Makefile需要严格按照这个名称，后面编译需要
```

可以在命令行输入 ls 查看是否正常创建文件，成功创建截图如下：

```
root@test:~/test1/padavan-ng/trunk/linux-3.4.x/drivers/ghykernel# ls
ghykernel.c  Kconfig  Makefile
```

3、接下来我们需要给三个文件分别写下对应的内容
①首先写入.c 文件：

```
vi ghykernel.c #进入文件的读写模式，同时回车后需要按下键盘上的"I"进入插入模式
```

.c 文件内容如下：

```
#include<linux/module.h>
MODULE_LICENSE("MIT");
MODULE_AUTHOR("GHY");
MODULE_DESCRIPTION("KERNEL EDIT EXAMINATION");
MODULE_VERSION("1.0");

//内核模块参数，加载时指定或者动态指定，以控制此模块的行为
static char *name ="GHY";
module_param(name,charp,S_IRUGO);
NODULE_PARAM_DESC(name,"---------now it is a print test---------");

//初始化函数，在加载时调用，分配资源准备执行环境
static int __init ghy_print_init(void){
    printk(KERN_INFO "TESTING: test case written by %s,this is kernel edit module\n",name);
    return 0;

}

//析构函数，在卸载时调用，回收资源，销毁执行环境
static void __exit ghy_print_exit(void){
    printk(KERN_INFO "TESTING: kernel test exit for %s \n",name);
}

//登记初始化函数和析构函
module_init(ghy_print_init);
module_exit(ghy_print_exit);

```

查看文件是否正确写入：

```
root@test:~/test1/padavan-ng/trunk/linux-3.4.x/drivers/ghykernel# cat ghykernel.c
#include<linux/module.h>
MODULE_LICENSE("MIT");
MODULE_AUTHOR("GHY");
MODULE_DESCRIPTION("KERNEL EDIT EXAMINATION");
MODULE_VERSION("1.0");

//内核模块参数，加载时指定或者动态指定，以控制此模块的行为
static char *name ="GHY";
module_param(name,charp,S_IRUGO);
NODULE_PARAM_DESC(name,"---------now it is a print test---------");

//初始化函数，在加载时调用，分配资源准备执行环境
static int __init ghy_print_init(void){
        printk(KERN_INFO "TESTING: test case written by %s,this is kernel edit module\n",name);
        return 0;

}

//析构函数，在卸载时调用，回收资源，销毁执行环境
static void __exit ghy_print_exit(void){
        printk(KERN_INFO "TESTING: kernel test exit for %s \n",name);
}

//登记初始化函数和析构函
module_init(ghy_print_init);
module_exit(ghy_print_exit);
```

②同样的步骤，写入 Kconfig 文件内容如下：

```
config GHY_KERNEL
        tristate "HELLO GHY_KERNEL"
        default y
```

③同样的步骤，写入 Makefile 文件内容如下：

```
obj-y += ghykernel.o
```

4、返回上一层目录，修改`./trunk/linux-3.4.x/drivers`目录下的 Kconfig 文件，操作如下，注意目录填自己刚刚第一步创建的文件名：

```
source "drivers/uio/Kconfig"

source "drivers/vlynq/Kconfig"

source "drivers/virtio/Kconfig"

source "drivers/hv/Kconfig"

source "drivers/xen/Kconfig"

source "drivers/staging/Kconfig"

source "drivers/platform/Kconfig"

source "drivers/clk/Kconfig"

source "drivers/hwspinlock/Kconfig"

source "drivers/clocksource/Kconfig"

source "drivers/iommu/Kconfig"

source "drivers/remoteproc/Kconfig"

source "drivers/rpmsg/Kconfig"

source "drivers/virt/Kconfig"

source "drivers/devfreq/Kconfig"

source "drivers/ghykernel/Kconfig"

endmenu
-- 插入 --                                                    145,35        底端
```

同理需要修改此目录下的 Makefile 文件：

3.3 编译一个应用模块

1、进入`./trunk/user`目录下，创建文件夹：

```
mkdir ghyapp
```

2、进入该目录下，创建两个文件.c 和 Makefile

```
touch main.c Makefile
```

3、在创建的文件中写入内容,注意内容涉及文件名的要用自己第一步创建的文件名
main.c 的内容：

```
#include<stdio.h>
int main(){
    printf("-----This is a TEST for APP----\n");
    //只要是c++在stdio.h支持下的代码都可以丢进去
    return 0;

}
```

Mekefile 内容如下：

```
CFLAGS += -ffunction-sections -fdata-sections -fPIC -std=gnu99
LDFLAGS += -Wl,--gc-sections

all : main.o Makefile
    $(CC) -o ghyapp main.o $(LDFLAGS)

main.o : main.c
    $(CC) -c main.c $(CFLAGS)

clean :
    rm -f main.o

clean-all :
    rm -f ghyapp *.o

romfs:
    $(ROMFSINST) ghyapp /bin/ghyapp

```

4、实验二和实验三一起编译，首先返回上层的`./trunk`目录，输入

```
./build_firmware.sh
```

编译成功结果如下：

```
make[2]: 进入目录“/root/test1/padavan-ng/trunk/vendors/Mediatek”
#############CREATE IMAGE################
/root/test1/padavan-ng/trunk/../toolchain/out/bin/mipsel-linux-uclibc-objcopy -O binary -R .note -R .comment -S /root/test1/padavan-ng
/trunk/linux-3.4.x/vmlinux /root/test1/padavan-ng/trunk/images/zImage
cd /root/test1/padavan-ng/trunk/images ; rm -f /root/test1/padavan-ng/trunk/images/zImage.*; \
/root/test1/padavan-ng/trunk/tools/lzma/lzma e -a1 -d25 /root/test1/padavan-ng/trunk/images/zImage /root/test1/padavan-ng/trunk/images
/zImage.lzma

LZMA 4.65 : Igor Pavlov : Public domain : 2009-02-03
# Padded Kernel Image Size
1153040 /root/test1/padavan-ng/trunk/images/zImage.lzma
# Original RootFs Size
25199168        /root/test1/padavan-ng/trunk/romfs
# Compressed RootFs Size
7882908 /root/test1/padavan-ng/trunk/images/ramdisk
# Padded Kernel Image + Compressed Rootfs Size
9035948 /root/test1/padavan-ng/trunk/images/zImage.lzma
#======================================
# Pack final image and write headers
img file: /root/test1/padavan-ng/trunk/images/YK-L1_3.4.3.9L-100.trx
Product ID:   YK-L1
Created:      Thu Nov 17 21:23:12 2022
Image Type:   MIPS Linux Kernel Image (lzma compressed)
Data Size:    9035948 Bytes = 8824.17 kB = 8.62 MB
Load Address: 0x80000000
Entry Point:  0x8029D760
Kernel Size:  0x00119850
Kernel Ver.:  3.4
FS Ver.:      3.9L
#======================================
Firmware size: 9036012
Firmware max allowed size: 32964608

make[2]: 离开目录“/root/test1/padavan-ng/trunk/vendors/Mediatek”
make[1]: 离开目录“/root/test1/padavan-ng/trunk/vendors”
root@test:~/test1/padavan-ng/trunk#
```

5、查找目标文件

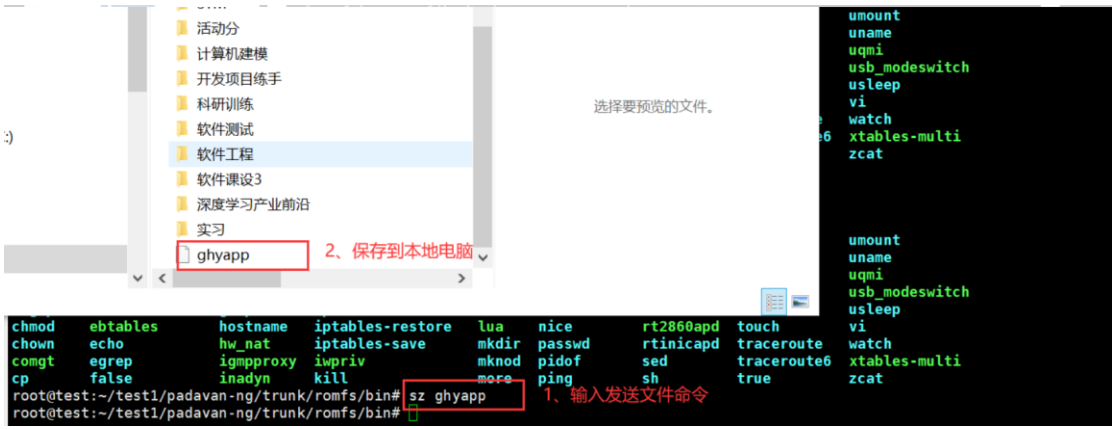
```
root@test:~/test1/padavan-ng/trunk# cd ./romfs/bin
root@test:~/test1/padavan-ng/trunk/romfs/bin# ls
ash      date           fgrep      ip                ldd     mount       ping6     sleep          umount
bash     dd             ghyapp     ip6tables         lld2d   mountpoint  ps        smbpasswd      uname
busybox  df             grep       ip6tables-restore ln      mtd_write   pwd       stat           uqmi
cat      dmesg          gunzip     ip6tables-save    login   mv          rm        sync           usb_modeswitch
chgrp    dnsdomainname  gzip       iptables          ls      netstat     rmdir     tar            usleep
chmod    ebtables       hostname   iptables-restore  lua     nice        rt2860apd touch          vi
chown    echo           hw_nat     iptables-save     mkdir   passwd      rtinicapd traceroute     watch
comgt    egrep          igmpproxy  iwpriv            mknod   pidof       sed       traceroute6    xtables-multi
cp       false          inadyn     kill              more    ping        sh        true           zcat
root@test:~/test1/padavan-ng/trunk/romfs/bin#
```

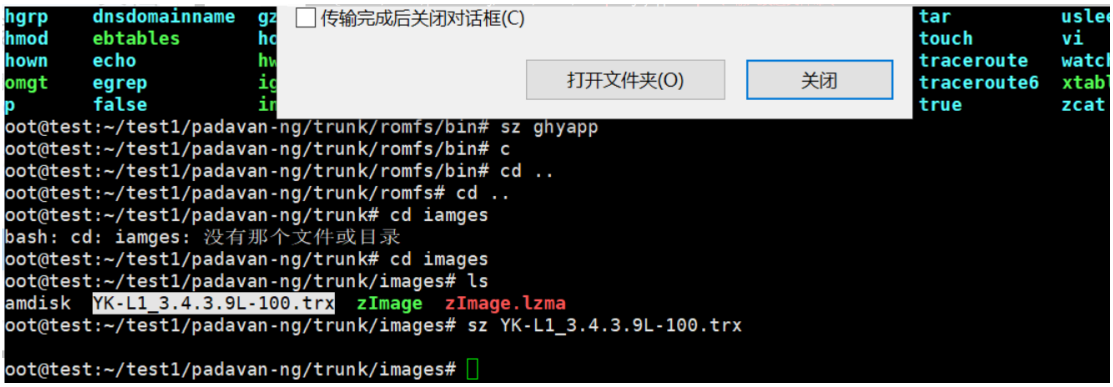
进入到./trunk/romfs/bin目录下

查看实验三生成的文件

- 进入`./trunk/romfs/bin`文件夹，查看自己实验三命名的 app 的 c 文件编译结构是否出现在这里
- 输入 rz [文件名]即可将结果保存至本机

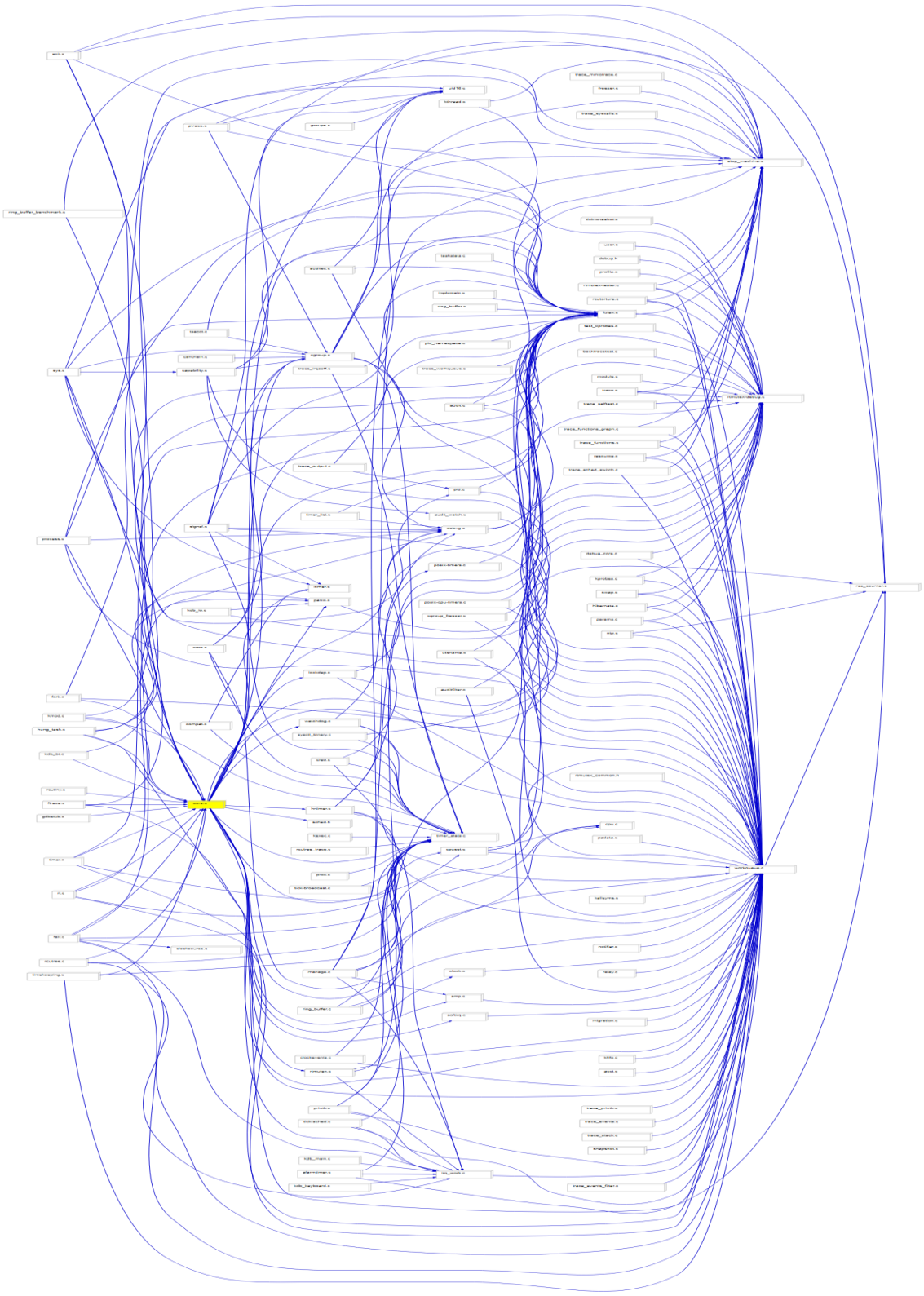
- 同理进入`./trunk/images`目录，将总的结果文件传输回来本机





3.4 core.c 注释和调度流程

本次注释的模块为\kernel\sched\core.c，具体注释见附件。
首先用 Understand 对代码进行静态分析，查看文件调用情况：



调度流程图：

主要的调度函数是__schedule()

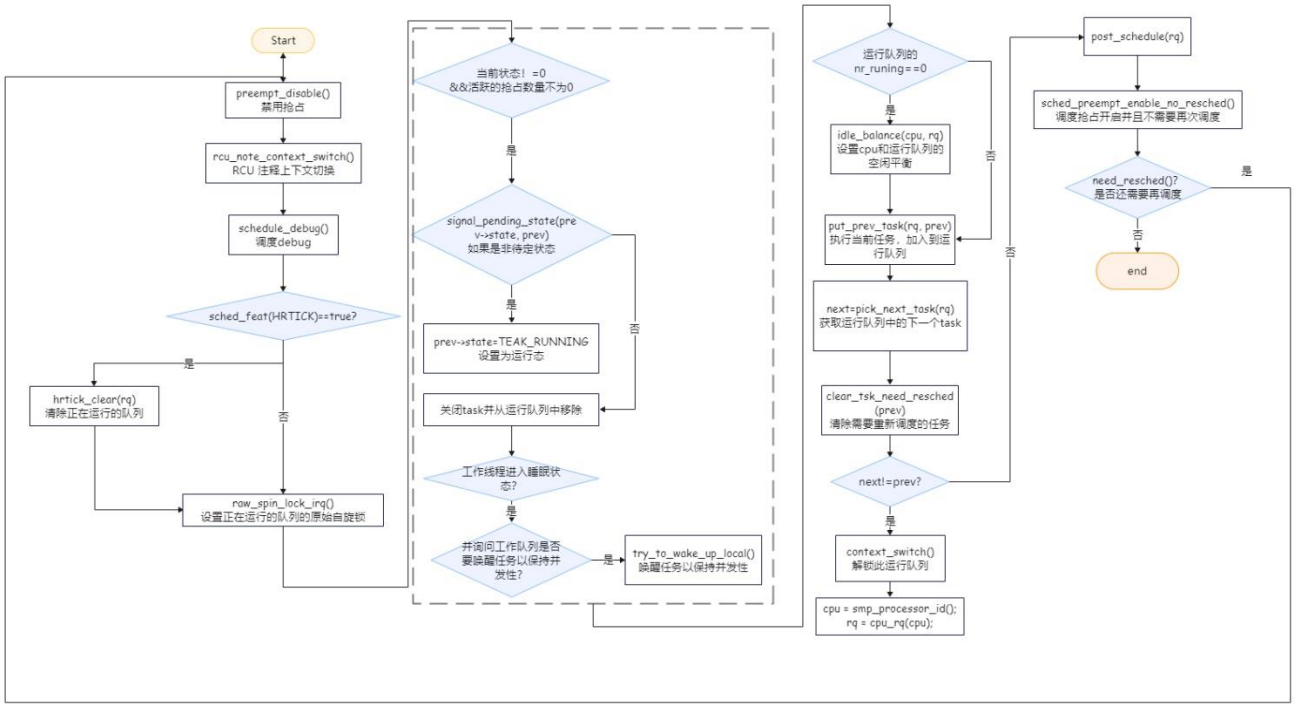
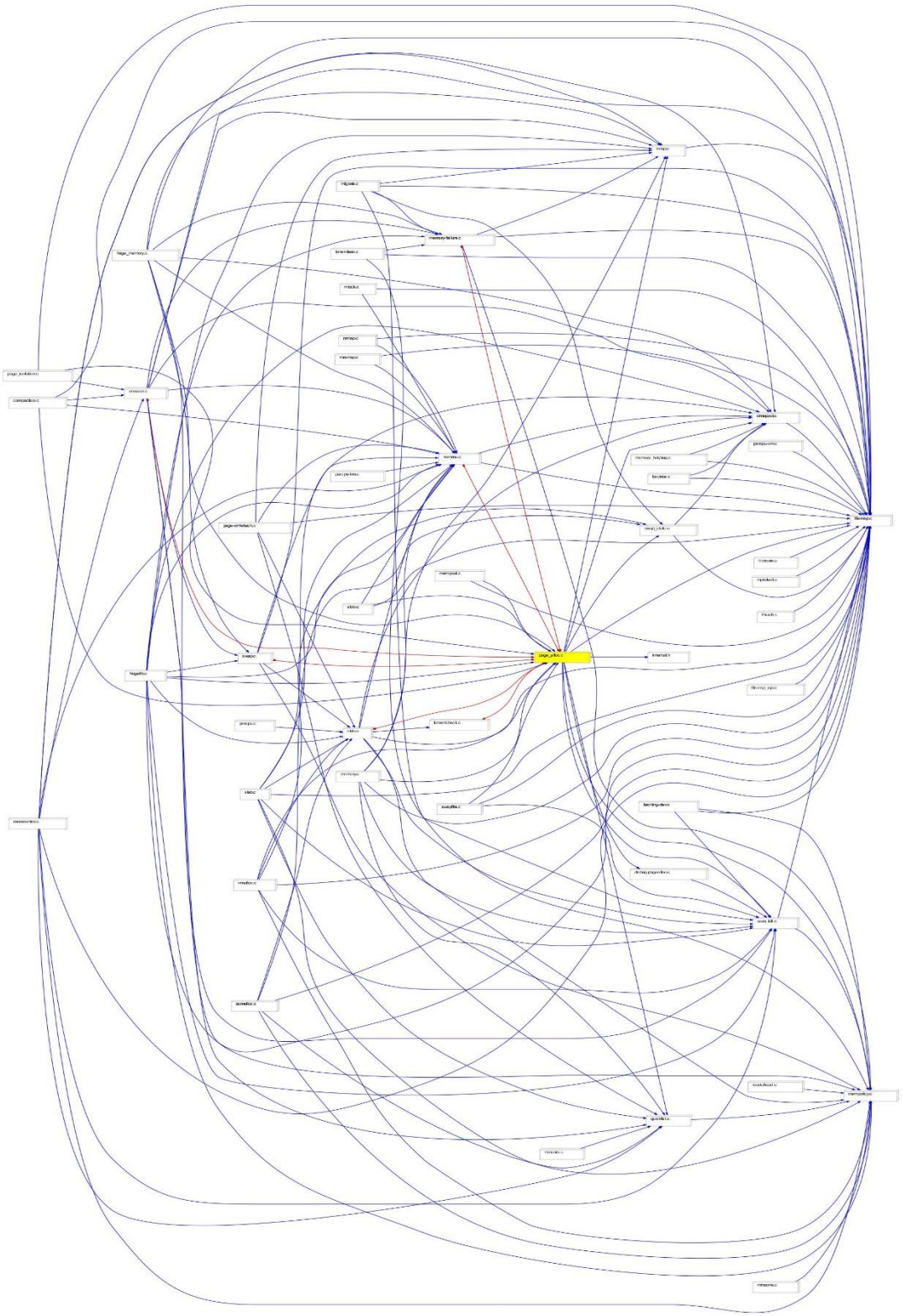


图 1 __schedule()函数流程图

3.5 page_alloc.c 注释和分配流程

本次注释的模块为 mm\page_alloc.c，具体注释见附件。
首先用 Understand 对代码进行静态分析，查看文件调用情况：



分配流程图：

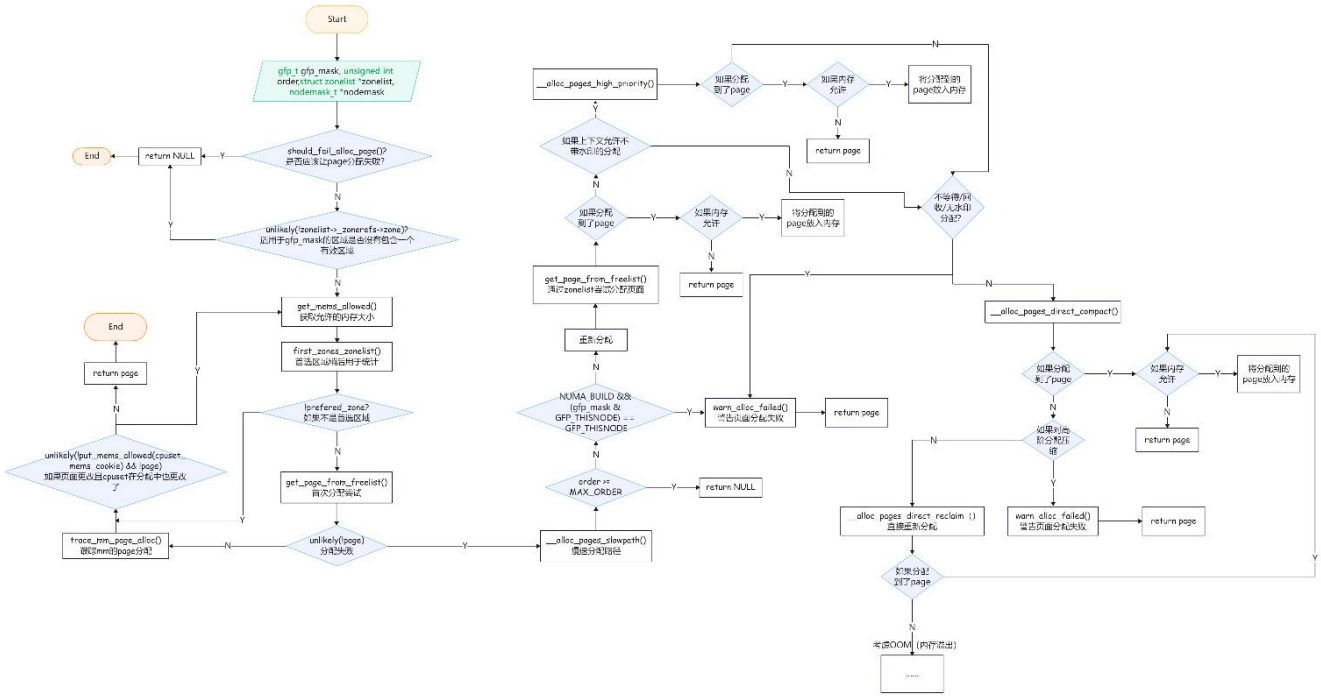


图 2 page_alloc 流程图

释放流程图：

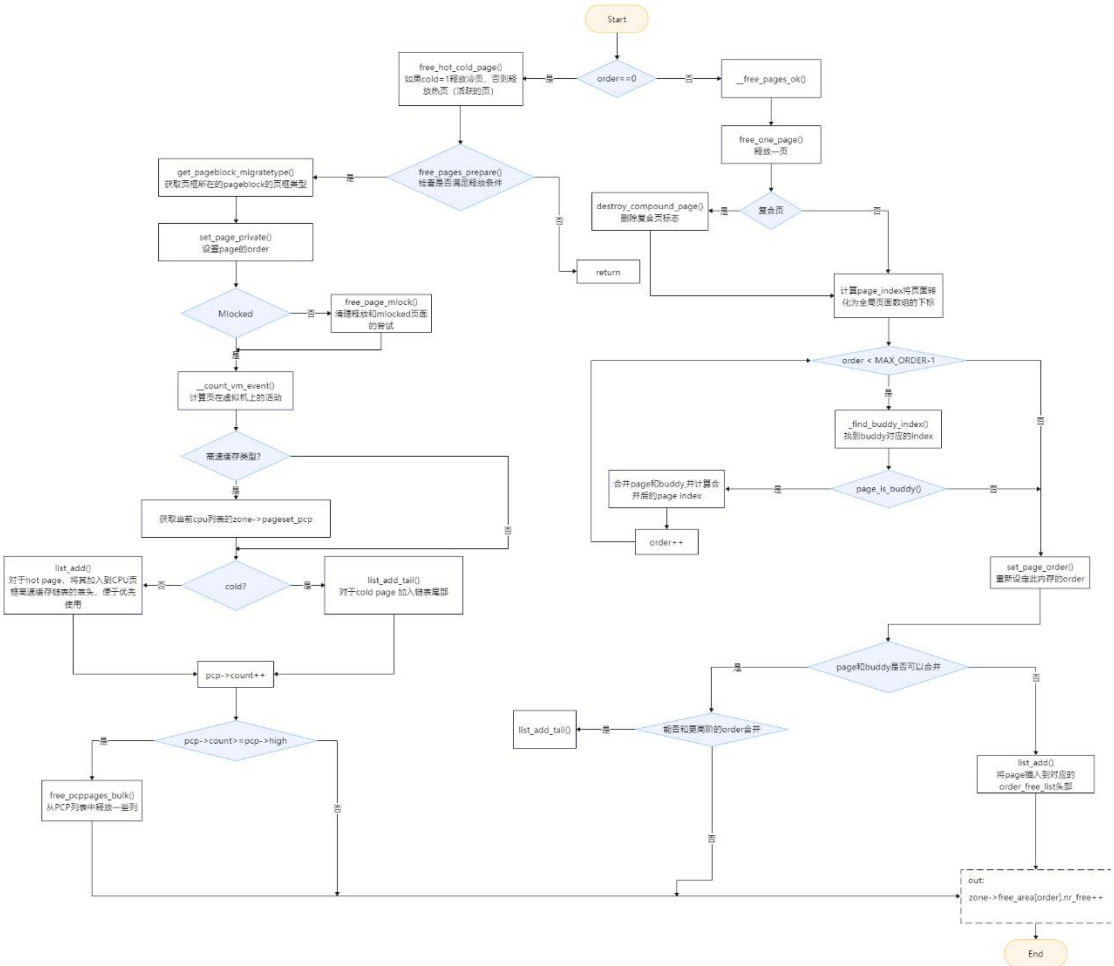


图 3 free_pages 流程图

4 项目总结

本次项目让我体会了 Linux 命令行编程，也算是为了之后的学习工作做了一个小小的铺垫。老师的项目指导手册非常清晰，诚如老师所言，代码的阅读理解上花费的时间最长，其实自己也不能完全确定自己的理解和流程图是否正确，读到这里希望老师没有被我自己写的流程图雷到。

11 月开始做这个课设，前三个任务完成的比较顺利，1-3 天耗时，主要被卡住的地方是下载 padavan-ng 项目，网速、科学上网、下载流量限制的坑挨个踩了一遍，花了两个下午的时间终于解决了网络和流量限制的问题（后来问同学发现好像只有我遇到了因为下载大小限制而造成的下载中断）。代码的注释是 11 月到 12 月期间断断续续做的，流程图也是在这个阶段用亿图图示慢慢画出来的。12 月后面受到疫情影响，很多课程结束的很匆忙，感觉自己很多事情没有安排好。保研之后有点迷茫，毕设、实习、开发项目——论忙碌，肯定是比不上大二大三，可能是自己还没有找准自己的定位吧，什么都想抓，但是什么也没有安排好。12 月在外面实习，正好阳了，在宾馆一个人躺了一周多，实习的进度落下了很多，正好赶上期末。这里很庆幸期末考核形式变得很亲民，现在写下这些的时候，我的身体已经恢复很多了也平安到家了，没有身体抱恙和外流浪的感觉，心情也变得阳光起来。

好像跑题了，拉回来，拉回来。

因为大四了，作为大学的最后一门课设，想浅浅谈一谈自己对软件课设 III 的看法：

首先这是一个非常棒的课设，不光让大家熟悉了虚拟机的使用，还以 server 的方式让大家体会远程使用虚拟机的工作流程，这很接近科研（比如开服务器来远程训练模型）以及一些实际工作场景（远程办公）。

选题上也非常有趣，padavan-ng 本身是一个非常有实际应用意义的项目，有很多手工爱好者自己下载、魔改、运行以及部署此项目来满足自身对于路由器一些功能的需要。

疫情原因，自己和同学购买的路由器最后没能带回家，所以项目没有实际刷写跑一跑看，算是一个小小的遗憾。

现在是元旦，写下这些内容内心非常充实，2022 已经过去，希望 2023 大家都身体健康，尽管疫情带走太多，仍然希望新的一年顺顺利利，万事大吉！

5 附录

- 附录一：core.c 以及 page_alloc.c 注释文件
- 附录二：流程图
- 附录三：任务一、二、三结果文件