操作系统实验报告

何宜晖
计算机46
2140504137
电信学院
heyihui@stu.xjtu.edu.cn

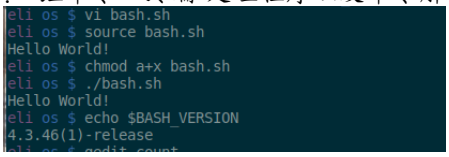
2016年11月

1 实验一:用户接口实验

为了使用户通过操作系统完成各项管理任务,操作系统必须为用户提供各种接口来实 现人机交互。经典的操作系统理论将操作系统的接口分为控制台命令和系统调用两种。前者 主要提供给计算机的操作人员对计算机进行各种控制;而后者则提供个程序员,使他们可以 方便地使用计算机的各种资源。

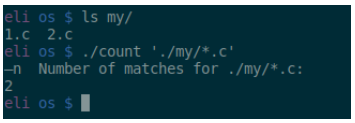
1.1 控制台命令接口

操作系统向用户提供一组控制台命令,用户可以通过终端输入命令的方式获得操作系统 的服务,并由此来控制自己作业的运行。一般来讲,控制台命令应该包含:一组命令、终端 处理程序以及命令解释程序



1.2 系统调用

1. 查看 bash 版本, 4.3.46
2. 编写 bash 脚本:统计/my 目录下 c 语言文件的个数



1.3 编程调用一个系统调用 fork()

```
eli os $ gedit fork.c
eli os $ ls
1.doc  a.c      b.c         fork.c     li
1.png  a.out    BNUCourses  fs         li
2.png  bash.sh  count       linux-3.1  li
eli os $ gcc fork.c -o fork.out
eli os $ ./fork.out
This is parent process.
This is child process.
This is parent process.
This is child process.
This is parent process.
This is child process.
This is parent process.
This is child process.
This is parent process.
This is child process.
This is child process.
This is parent process.
This is parent process.
This is child process.
This is parent process.
This is child process.
This is child process.
This is parent process.
This is parent process.
This is child process.
This is child process.
This is parent process.
This is child process.
This is parent process.
This is parent process.
This is child process.
This is child process.
This is parent process.
This is child process.
This is parent process.
This is parent process.
This is child process.
This is parent process.
This is child process.
This is child process.
```

1.4 kernel 编译

1.4.1 文件准备

ubuntu version: 16.10, kernel version: 4.8.9

```
eli os $ lsb_release -a
LSB Version:    core-2.0-amd64:core-2.0-noarch:core-3.0-amd64:core-3.0-noarch:core-
noarch:core-4.1-amd64:core-4.1-noarch
Distributor ID: Ubuntu
Description:    Ubuntu Zesty Zapus (development branch)
Release:        17.04
Codename:       zesty
eli os $ uname -a
Linux eli 4.8.9 #1 SMP Sun Nov 20 15:58:32 CST 2016 x86_64 x86_64 x86_64 GNU/Linux
eli os $
```

首先在 http://kernel.org 上面下载需要编译的版本的内核。下载完成的源码包,为*.tar.xz 的格式。在任意目录解压源码包。

```
eli os $ ls linux-4.8.9*
linux-4.8.9.tar.xz

linux-4.8.9:
arch           firmware  lib              newcall          tools
block          fs        MAINTAINERS      README           usr
certs          include   Makefile         REPORTING-BUGS   virt
COPYING        init      mm               samples          vmlinux
CREDITS        ipc       modules.builtin  scripts          vmlinux-gdb.py
crypto         Kbuild    modules.order    security         vmlinux.o
Documentation  Kconfig   Module.symvers   sound
drivers        kernel    net              System.map
eli os $
```

1.4.2 依赖包准备

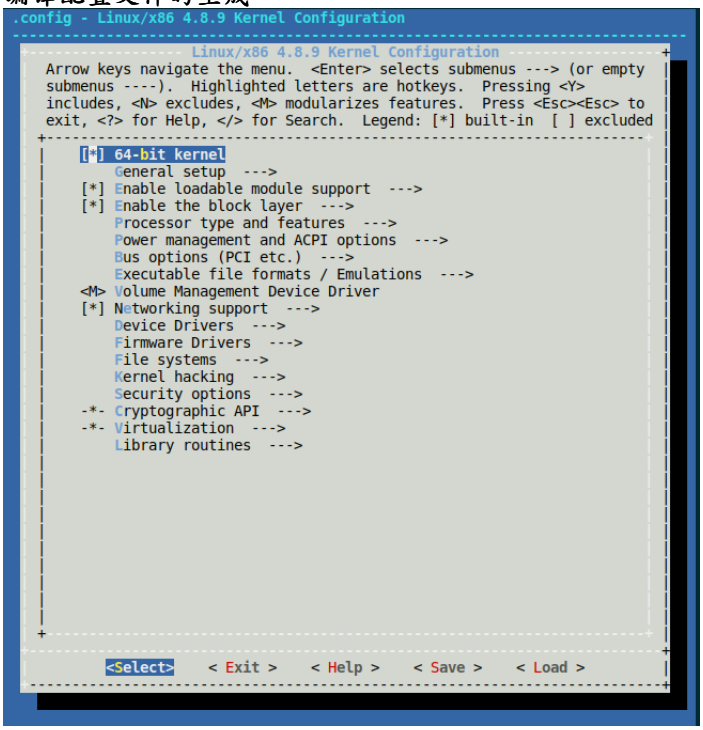
更新软件列表,安装必备组件. 系统自带的是g++ 6.2, 需要降级至 5.4 . 否则后面编译会出问题。

```
ell os $ g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/6/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 6.2.0-7ubuntu11' -
-with-bugurl=file:///usr/share/doc/gcc-6/README.Bugs --enable-languages=c,ada,c++
,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-6 --program-prefi
x=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/li
b --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nl
s --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdc
xx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable
-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system
-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-ja
va-home=/usr/lib/jvm/java-1.5.0-gcj-6-amd64/jre --enable-java-home --with-jvm-roo
t-dir=/usr/lib/jvm/java-1.5.0-gcj-6-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports
/java-1.5.0-gcj-6-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/jav
a/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arc
h-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --wi
th-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-
linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 6.2.0 20161018 (Ubuntu 6.2.0-7ubuntu11)
```

若使用 menuconfig 生成配置文件,首先要安装相关的依赖库。 我使用menuconfig。

1.4.3 编译过程

编译配置文件的生成



开始编译

一开始编译失败，原因是使用了gcc6,切换至gcc5后正常，先make clean，再make。

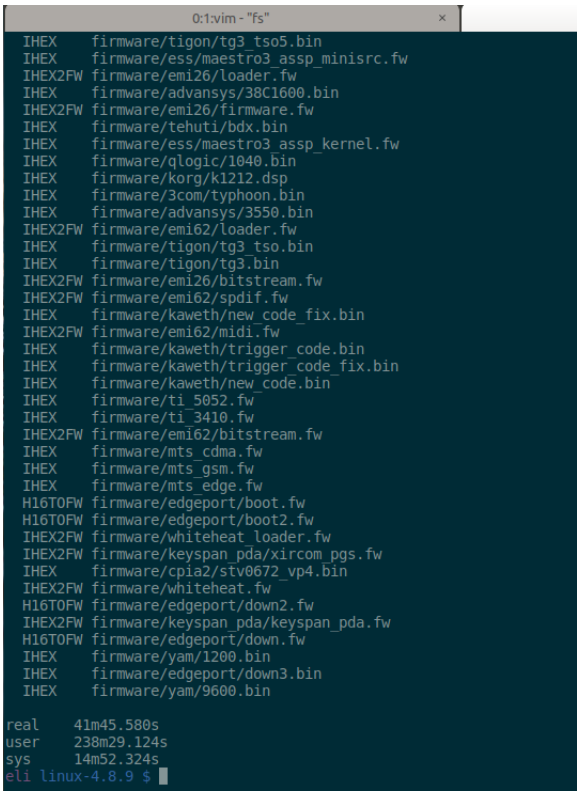
```
eli linux-4.8.9 $ make clean
  CLEAN   .
  CLEAN   arch/x86/entry/vdso
  CLEAN   arch/x86/kernel/cpu
  CLEAN   arch/x86/kernel
  CLEAN   arch/x86/purgatory
  CLEAN   arch/x86/realmode/rm
  CLEAN   arch/x86/lib
  CLEAN   certs
  CLEAN   crypto/asymmetric_keys
  CLEAN   crypto
  CLEAN   drivers/firmware/efi/libstub
  CLEAN   drivers/gpu/drm/radeon
  CLEAN   drivers/net/wan
  CLEAN   drivers/scsi/aic7xxx
  CLEAN   drivers/tty/vt
  CLEAN   firmware
  CLEAN   kernel/debug/kdb
  CLEAN   lib/raid6
  CLEAN   lib
  CLEAN   security/apparmor
  CLEAN   security/selinux
  CLEAN   security/tomoyo
  CLEAN   usr
  CLEAN   arch/x86/boot/compressed
  CLEAN   arch/x86/boot
  CLEAN   arch/x86/tools
  CLEAN   .tmp_versions
```

在确保.config 文件已经正确生成的前提之下,就可以开始编译内核了(可以使用-j 参数 加速编译过程)

CODE:make

```
eli linux-4.8.9 $ time make -j17
  CHK     include/config/kernel.release
  HOSTCC  scripts/basic/fixdep
  CHK     include/generated/uapi/linux/version.h
  HOSTCC  scripts/basic/bin2c
  HOSTCC  arch/x86/tools/relocs_32.o
  CHK     include/generated/utsrelease.h
  HOSTCC  arch/x86/tools/relocs_64.o
  HOSTCC  arch/x86/tools/relocs_common.o
  HOSTCC  scripts/conmakehash
  HOSTCC  scripts/kallsyms
  HOSTCC  scripts/sortextable
  HOSTCC  scripts/recordmcount
  HOSTCC  scripts/asn1_compiler
  HOSTCC  scripts/extract-cert
  CC      scripts/mod/empty.o
  HOSTCC  scripts/insert-sys-cert
  HOSTCC  scripts/sign-file
  HOSTCC  scripts/genksyms/parse.tab.o
  HOSTCC  scripts/genksyms/genksyms.o
  HOSTCC  scripts/selinux/genheaders/genheaders
  HOSTCC  scripts/selinux/mdp/mdp
  HOSTCC  scripts/mod/mk_elfconfig
  CC      scripts/mod/devicetable-offsets.s
  HOSTCC  scripts/genksyms/lex.lex.o
  GEN     scripts/mod/devicetable-offsets.h
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/sumversion.o
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTLD  arch/x86/tools/relocs
  CC      arch/x86/purgatory/string.o
  CC      arch/x86/purgatory/purgatory.o
  AS      arch/x86/purgatory/setup-x86_64.o
  AS      arch/x86/purgatory/stack.o
  CC      arch/x86/purgatory/sha256.o
  AS      arch/x86/purgatory/entry64.o
  HOSTLD  scripts/genksyms/genksyms
  LD      arch/x86/purgatory/purgatory.ro
  BIN2C   arch/x86/purgatory/kexec-purgatory.c
```

编译完成



安装模块和内核

由于版本差异在 3.10 版本以上的内核编译时首先进行模块安装

CODE: sudo make modules_install

```
eli linux-4.8.9 $ sudo make modules_install
[sudo] password for eli:
  INSTALL arch/x86/crypto/aes-x86_64.ko
  INSTALL arch/x86/crypto/aesni-intel.ko
  INSTALL arch/x86/crypto/blowfish-x86_64.ko
  INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
  INSTALL arch/x86/crypto/camellia-aesni-avx2.ko
  INSTALL arch/x86/crypto/camellia-x86_64.ko
  INSTALL arch/x86/crypto/cast5-avx-x86_64.ko
  INSTALL arch/x86/crypto/cast6-avx-x86_64.ko
  INSTALL arch/x86/crypto/chacha20-x86_64.ko
  INSTALL arch/x86/crypto/crc32-pclmul.ko
  INSTALL arch/x86/crypto/crct10dif-pclmul.ko
  INSTALL arch/x86/crypto/des3_ede-x86_64.ko
  INSTALL arch/x86/crypto/ghash-clmulni-intel.ko
  INSTALL arch/x86/crypto/glue_helper.ko
  INSTALL arch/x86/crypto/poly1305-x86_64.ko
  INSTALL arch/x86/crypto/salsa20-x86_64.ko
  INSTALL arch/x86/crypto/serpent-avx-x86_64.ko
  INSTALL arch/x86/crypto/serpent-avx2.ko
  INSTALL arch/x86/crypto/serpent-sse2-x86_64.ko
  INSTALL arch/x86/crypto/sha1-mb/sha1-mb.ko
  INSTALL arch/x86/crypto/sha1-ssse3.ko
  INSTALL arch/x86/crypto/sha256-mb/sha256-mb.ko
  INSTALL arch/x86/crypto/sha256-ssse3.ko
  INSTALL arch/x86/crypto/sha512-mb/sha512-mb.ko
  INSTALL arch/x86/crypto/sha512-ssse3.ko
  INSTALL arch/x86/crypto/twofish-avx-x86_64.ko
  INSTALL arch/x86/crypto/twofish-x86_64-3way.ko
  INSTALL arch/x86/crypto/twofish-x86_64.ko
  INSTALL arch/x86/events/intel/intel-cstate.ko
  INSTALL arch/x86/events/intel/intel-rapl-perf.ko
  INSTALL arch/x86/kernel/cpu/mcheck/mce-inject.ko
  INSTALL arch/x86/kernel/cpuid.ko
  INSTALL arch/x86/kernel/msr.ko
  INSTALL arch/x86/kvm/kvm-amd.ko
  INSTALL arch/x86/kvm/kvm-intel.ko
  INSTALL arch/x86/kvm/kvm.ko
  INSTALL /lib/firmware/edgeport/down2.fw
  INSTALL /lib/firmware/edgeport/down3.bin
  INSTALL /lib/firmware/whiteheat_loader.fw
  INSTALL /lib/firmware/whiteheat.fw
  INSTALL /lib/firmware/keyspan_pda/keyspan_pda.fw
  INSTALL /lib/firmware/keyspan_pda/xircom_pgs.fw
  INSTALL /lib/firmware/cpia2/stv0672_vp4.bin
  INSTALL /lib/firmware/yam/1200.bin
  INSTALL /lib/firmware/yam/9600.bin
  DEPMOD  4.8.9
eli linux-4.8.9 $ C
```

之后再进行内核安装, 在 3.10 版本以上的内核安装时,会自动进行 initrd 的生成以及 GRUB 的更新

CODE: sudo make install

```
eli linux-4.8.9 $ sudo make install
sh ./arch/x86/boot/install.sh 4.8.9 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.8.9 /boot/vmlinuz-4.8.9
run-parts: executing /etc/kernel/postinst.d/dkms 4.8.9 /boot/vmlinuz-4.8.9
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.8.9 /boot/vmlinuz-4.8.9
update-initramfs: Generating /boot/initrd.img-4.8.9
W: Possible missing firmware /lib/firmware/i915/kbl_guc_ver9_14.bin for module i915
W: Possible missing firmware /lib/firmware/i915/bxt_guc_ver8_7.bin for module i915
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.8.9 /boot/vmlinuz-4.8.9
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.8.9 /boot/vmlinuz-4.8.9
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.8.9 /boot/vmlinuz-4.8.9
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.8.9 /boot/vmlinuz-4.8.9
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no lon
Found linux image: /boot/vmlinuz-4.8.9
Found initrd image: /boot/initrd.img-4.8.9
Found linux image: /boot/vmlinuz-4.8.9.old
Found initrd image: /boot/initrd.img-4.8.9
Found linux image: /boot/vmlinuz-4.8.0-27-generic
Found initrd image: /boot/initrd.img-4.8.0-27-generic
Found linux image: /boot/vmlinuz-4.8.0-26-generic
Found initrd image: /boot/initrd.img-4.8.0-26-generic
Found linux image: /boot/vmlinuz-4.4.0-45-generic
Found initrd image: /boot/initrd.img-4.4.0-45-generic
Found linux image: /boot/vmlinuz-4.2.0-35-generic
Found initrd image: /boot/initrd.img-4.2.0-35-generic
Adding boot menu entry for EFI firmware configuration
done
eli linux-4.8.9 $ C
```

完成之后直接重启,之后查看 uname

1.4.4 使用新内核启动

查看新系统的内核版本

```
eli os $ lsb_release -a
LSB Version:    core-2.0-amd64:core-2.0-noarch:core-3.0-amd64:core-3.0-noarch:core-
noarch:core-4.1-amd64:core-4.1-noarch
Distributor ID: Ubuntu
Description:    Ubuntu Zesty Zapus (development branch)
Release:        17.04
Codename:       zesty
eli os $ uname -a
Linux eli 4.8.9 #1 SMP Sun Nov 20 15:58:32 CST 2016 x86_64 x86_64 x86_64 GNU/Linux
eli os $
```

1.4.5 添加 System_call

本实验基于上个实验,首先要保证能够正常进入自己编译的内核之后。在做此实验。

源文件的更改

```
eli linux-4.8.9 $ ls
arch           firmware  lib             newcall         tools
block          fs        MAINTAINERS     README          usr
certs          include   Makefile        REPORTING-BUGS  virt
COPYING        init      mm              samples         vmlinux
CREDITS        ipc       modules.builtin scripts         vmlinux-gdb.py
crypto         Kbuild    modules.order   security        vmlinux.o
Documentation  Kconfig   Module.symvers  sound
drivers        kernel    net             System.map
eli linux-4.8.9 $ ls newcall/
built-in.o  Makefile  modules.builtin  modules.order  newcall.c  newcall.o
eli linux-4.8.9 $
```

编写新的系统调用,编写编译配置文件

```
eli newcall $ cat Makefile
obj-y := newcall.o
eli newcall $ cat newcall.c
#include<linux/linkage.h>
asmlinkage long sys_newcall(int i){
    return (i*10);
}
eli newcall $
```

添加系统调用入口

4.8.9 调用入口位置不太一样， 在

```
vi x86/entry/syscalls/syscall_64.tbl
```

```
331 322 64  execveat        sys_execveat/ptregs
332 323 common  userfaultfd     sys_userfaultfd
333 324 common  membarrier      sys_membarrier
334 325 common  mlock2          sys_mlock2
335 326 common  copy_file_range     sys_copy_file_range
336 327 64  preadv2         sys_preadv2
337 328 64  pwritev2        sys_pwritev2
338 329 64  mmsyscal        sys_newcall
339
340 #
341 # x32-specific system call numbers start at 512 to avoid cache im
342 # for native 64-bit operation.
343 #
344 512 x32 rt_sigaction        compat_sys_rt_sigaction
345 513 x32 rt_sigreturn         sys32_x32_rt_sigreturn
346 514 x32 ioctl           compat_sys_ioctl
347 515 x32 readv           compat_sys_readv
348 516 x32 writev          compat_sys_writev
349 517 x32 recvfrom        compat_sys_recvfrom
```

修改整体调用

修改文件/include/linux/syscalls.h,在这个文件的最后按照格式添加上自己的系统调用。

```
881 asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
882 ›   ›   ›   unsigned long idx1, unsigned long idx2);
883 asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags#
884 asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
885 ›   ›   ›   const char __user *uargs);
886 asmlinkage long sys_getrandom(char __user *buf, size_t count,
887 ›   ›   ›   unsigned int flags);
888 asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);
889
890 asmlinkage long sys_execveat(int dfd, const char __user *filename,
891 ›   ›   ›   const char __user *const __user *argv,
892 ›   ›   ›   const char __user *const __user *envp, int flags);
893
894 asmlinkage long sys_membarrier(int cmd, int flags);
895 asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
896 ›   ›   ›   ›   int fd_out, loff_t __user *off_out,
897 ›   ›   ›   ›   size_t len, unsigned int flags);
898
899 asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
900
901 asmlinkage long sys_newcall(int i);
902
903 #endif
 NORMAL > SPELL ›  1:syscalls.h          cpp  100% : 903/903 :  1  ! mixed-i…
neocomplete requires Vim 7.3.885 or later with Lua support ("+lua").
```

添加新系统调用至内核编译的配置文件

更改 kernel 编译的 Makefile,添加进自己的系统调用。更改所示的那一行,添加入自己所编写的系统调用的文件夹。

```
882 ifdef CONFIG_MODULE_SIG_ALL
883 $(eval $(call config_filename,MODULE_SIG_KEY))
884
885 mod_sign_cmd = scripts/sign-file $(CONFIG_MODULE_SIG_HASH) $(MODULE_SIG_KEY#
886 else
887 mod_sign_cmd = true
888 endif
889 export mod_sign_cmd
890
891
892 ifeq ($(KBUILD_EXTMOD),)
893 core-y› ›   += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ newcall/
894
895 vmlinux-dirs›   := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
896 ›   ›   ›   $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
897 ›   ›   ›   $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))
898
899 vmlinux-alldirs›:= $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
900 ›   ›   ›   $(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))
901
```

1.4.6 编译更改之后的内核源码

```
eli linux-4.8.9 $ time make -j17
  CHK     include/config/kernel.release
  HOSTCC  scripts/basic/fixdep
  CHK     include/generated/uapi/linux/version.h
  HOSTCC  scripts/basic/bin2c
  HOSTCC  arch/x86/tools/relocs_32.o
  CHK     include/generated/utsrelease.h
  HOSTCC  arch/x86/tools/relocs_64.o
  HOSTCC  arch/x86/tools/relocs_common.o
  HOSTCC  scripts/conmakehash
  HOSTCC  scripts/kallsyms
  HOSTCC  scripts/sortextable
  HOSTCC  scripts/recordmcount
  HOSTCC  scripts/asn1_compiler
  HOSTCC  scripts/extract-cert
  CC      scripts/mod/empty.o
  HOSTCC  scripts/insert-sys-cert
  HOSTCC  scripts/sign-file
  HOSTCC  scripts/genksyms/parse.tab.o
  HOSTCC  scripts/genksyms/genksyms.o
  HOSTCC  scripts/selinux/genheaders/genheaders
  HOSTCC  scripts/selinux/mdp/mdp
  HOSTCC  scripts/mod/mk_elfconfig
  CC      scripts/mod/devicetable-offsets.s
  HOSTCC  scripts/genksyms/lex.lex.o
  GEN     scripts/mod/devicetable-offsets.h
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/sumversion.o
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTLD  arch/x86/tools/relocs
  CC      arch/x86/purgatory/string.o
  CC      arch/x86/purgatory/purgatory.o
  AS      arch/x86/purgatory/setup-x86_64.o
  AS      arch/x86/purgatory/stack.o
  CC      arch/x86/purgatory/sha256.o
  AS      arch/x86/purgatory/entry64.o
  HOSTLD  scripts/genksyms/genksyms
  LD      arch/x86/purgatory/purgatory.ro
  BIN2C   arch/x86/purgatory/kexec-purgatory.c
```

1.4.7 测试新的 System_call

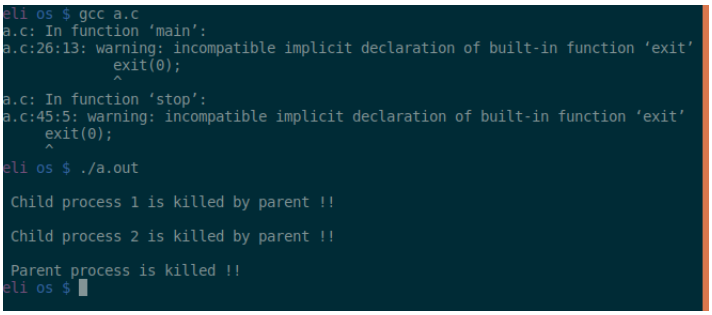
```
eli os $ cat test.c
#include<stdio.h>
#include<linux/unistd.h>

long newcall(int i){
    return syscall(329,i);
}

int main(void){
    printf("%ld\n", newcall(10));
    return 0;
}
eli os $ gcc test.c
eli os $ ./a.out
100
eli os $
```

2 实验二:进程管理

系统调用是一种进入系统空间的办法。通常,在 OS 的核心中都设置了一组用于实现各 种系统功能的子程序, 并将他们提供给程序员使用。 程序员在需要 OS 提供某种服务的时候, 便可以调用一条系统调用命令,去实现希望的功能,这就是系统调用。因此,系统调用就像 一个黑箱子一样,对用户屏蔽了操作系统的具体动作而只是提供了调用功能的接口。

调用fork时，返回值为0表示子进程，返回值大于0表示父进程，小于0表示调用失败。

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
int wait_flag;
void stop();
main()
{
    int pid1, pid2;
    signal(3, stop);
    while ((pid1 = fork()) == -1)
        ;
    if (pid1 > 0)
    {
        while ((pid2 = fork()) == -1)
            ;
        if (pid2 > 0)
        {
            wait_flag = 1;
            sleep(5);
            kill(pid1, 16);
            kill(pid2, 17);
            wait(0);
            wait(0);
            printf("\n Parent process is killed !!\n");
            exit(0);
        }
        else
        {
            wait_flag = 1;
            signal(17, stop);
            printf("\n Child process 2 is killed by parent !!\n");
        }
    }
    else
    {
        wait_flag = 1;
        signal(16, stop);
        printf("\n Child process 1 is killed by parent !!\n");
    }
}
void stop()
{
    wait_flag = 0;
    exit(0);
}
```



```c
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
int pid1, pid2;
main()
{
    int fd[2];
    char OutPipe[100], InPipe[100];
    pipe(fd);
    while ((pid1 = fork()) == -1);
    if (pid1 == 0)
    {
        lockf(fd[1], 1, 0);
        sprintf(OutPipe, "\n Child process 1 is sending message!\n");
        write(fd[1], OutPipe, 50);
        sleep(5);
        lockf(fd[1], 0, 0);
        exit(0);
    }
    else
    {
        while ((pid2 = fork()) == -1)
            ;
        if (pid2 == 0)
        {
            lockf(fd[1], 1, 0);
            sprintf(OutPipe, "\n Child process 2 is sending message!\n");
            write(fd[1], OutPipe, 50);
            sleep(5);
            lockf(fd[1], 0, 0);
            exit(0);
        }
        else
        {
            wait(0);
            read(fd[0], InPipe, 50);
            printf("%s\n", InPipe);
            wait(0);
```

```
            read(fd[0], InPipe, 50);
            printf("%s\n", InPipe);
            exit(0);
        }
    }
}
```

```
eli os $ gcc b.c
b.c: In function 'main':
b.c:18:9: warning: incompatible implicit declaration of built-in function 'exit'
        exit(0);
        ^
b.c:31:13: warning: incompatible implicit declaration of built-in function 'exit'
            exit(0);
            ^
b.c:41:13: warning: incompatible implicit declaration of built-in function 'exit'
            exit(0);
            ^
eli os $ ./a.out

 Child process 1 is sending message!


 Child process 2 is sending message!
```

3 实验三:存储器管理实验

本实验并没有进入系统空间对实际进程页面进行控制,而是在用户空间用线性表的连续 存储方式对进程页面交换进行模拟。

实现FIFO，LRU，NUR，OPT算法。

```
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <stdio.h>
#include <string>

using namespace std;

#define total_instruction 10
#define M 10 // pages

#define N 3 // available pages
struct Pro
{
    int num, time;
};
int page[N];

void print(Pro *page1)
// print current page
{
    Pro *page = new Pro[N];
    page = page1;
    for (int i = 0; i < N; i++)
```

```
   printf("%-4d", page[i].num);
    cout << endl;
}

int Search(int e, Pro *page1)
{
    Pro *page = new Pro[N];
    page = page1;
    for (int i = 0; i < N; i++)
   if (e == page[i].num)
       return i;
    return -1;
}

int Searchtime(int e, Pro *page1)
{
    Pro *page = new Pro[N];
    page = page1;
    for (int i = 0; i < N; i++)
   if (e == page[i].time)
       return i;
    return -1;
}

int Max(Pro *page1)
{
    Pro *page = new Pro[N];
    page = page1;
    int e = page[0].time, i = 0;
    while (i < N)
    //longest
    {
   if (e < page[i].time)
       e = page[i].time;
   i++;
    }
    for (i = 0; i < N; i++)
   if (e == page[i].time)
       return i;

    return -1;
}

int Compfu(Pro *page1, int i, int t, Pro p[M])
{
    Pro *page = new Pro[N];
    page = page1;

    int count = 0;
    for (int j = i; j < M; j++)
```

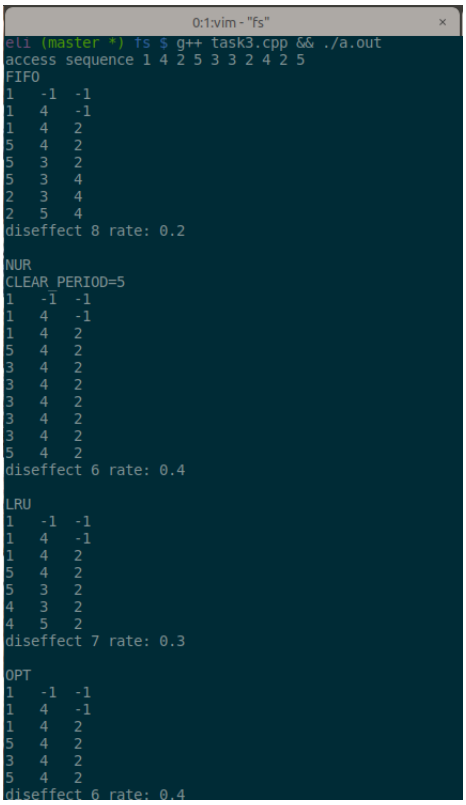
```
    {
   if (page[t].num == p[j].num)
      break;
   else
      count++;
    }
    return count;
}

int main()
{
    Pro p[total_instruction];
    Pro *page = new Pro[N];
    int t = 0, i, algo;
    float n = 0;
    int a[total_instruction] = {1, 4, 2, 5, 3, 3, 2, 4, 2, 5};
    printf("access sequence ");
    for (i = 0; i < total_instruction; i++)
    {
   p[i].num = a[i];
   cout << a[i] << " ";
    }

    for (algo = 0; algo < 4; algo++)
    {

   for (i = 0; i < N; i++) //init
   {
      page[i].num = -1;
      page[i].time = 2 - i;
   }

   cout << endl;
   i = 0;

   if (algo == 0)
   {
      cout << "FIFO" << endl;
      n = 0;

      while (i < total_instruction)
      {
     if (Search(p[i].num, page) >= 0)
        // found the page in memory
        i++;
     else
     {
        if (t == N)
       t = 0;
        else
```

```
        {
       n++; //
       page[t].num = p[i].num;
       print(page);
       t++;
        }
    }
     }
}
if (algo == 1)
{
    cout << "NUR" << endl;

    n = 0;
    cout << "CLEAR_PERIOD=5" << endl;
    int period = 0;
    int time_set;
    while (i < total_instruction)
    {
   if (period % 10 == 0)
   {
       for (int q = 0; q < N; q++)
      page[q].time = 0;
   }
   t = Search(p[i].num, page);
   if (t >= 0)
   {
       page[t].time = 1;
   }
   else
   {
       time_set = Searchtime(0, page);
       if (time_set == -1)
       {
      page[0].num = p[i].num;
      n++;
       }
       else
       {
      page[time_set].num = p[i].num;
      page[time_set].time = 1;
      n++;
       }
   }
   print(page);
   i++;
   period++;
    }
}
```

```
if (algo == 2)
{
    cout << "LRU" << endl;
    n = 0;
    while (i < total_instruction)
    {
   int k;
   k = t = Search(p[i].num, page);
   if (t >= 0)
       page[t].time = 0;
   else
   {
       n++;
       t = Max(page);
       page[t].num = p[i].num;
       page[t].time = 0;
   }
   for (int j = 0; j < N; j++)
   {
       if (j != t)
      page[j].time++;
   }
   if (k == -1)
       print(page);
   i++;
    }
}
if (algo == 3)
{
    cout << "OPT" << endl;
    n = 0;
    while (i < total_instruction)
    {
   if (Search(p[i].num, page) >= 0)
       i++;
   else
   {
       if (page[N - 1].num == -1)
       {
      for (int g = 0; g < N; g++)
          if (page[g].num == -1)
          {
         page[g].num = p[i].num;
         i++;
         n++;
         print(page);
         break;
          }
       }
       else
```

```
        {
        int temp = -1, cn;
        for (t = 0; t < N; t++)
        {
            if (temp < Compfu(page, i, t, p))
            {
           temp = Compfu(page, i, t, p);
           cn = t;
            }
        }
        page[cn] = p[i];
        n++;
        print(page);
        i++;
         }
      }
       }
   }
   cout << "diseffect " << n << " rate: " << 1 - n / total_instruction
       << endl;
    }
    return 0;
}
```

访问序列使用实验指导书上的例子, 运行结果：



4 实验四:文件系统实验

这是相对来说有一定难度的实验,它含盖了一个简单的二级文件系统的设计以及相关的 接口命令编写的内容,也鉴于此把它放在了最后一个实验。“一分耕耘,一分收获”,在完 整的完成本实验,你将获得的收益是:对文件系统工作的机理,特别是 linux 的 ext2 文件 系统工作机理了如指掌;linux 下较强的编程能力。好了,从此开始:

A 样例代码与输出