نام و نام خانوادگی: حامد جانزاده، وحیدکاظم‌پور	*تاریخ: 21/3/84*	
نام کار: مطالعه بر روی ساختار Live CD و بررسی فایل‌سیستم‌های squashfs و unionfs	نگارش: ۱.۰	

تاریخچه نگارش

نویسنده	توضیح	بررسی‌کننده	نگارش	تاریخ
حامد جانزاده - وحید کاظم‌پور	طالعه بر روی ساختار Live CD و بررسی فایل‌سیستم‌های squashfs و unionfs	امیرحسین پی‌براه	۱.۰	84/3/21

نام و نام خانوادگی: حامد جانزاده، وحیدکاظم‌پور	*تاریخ: 21/3/84*	
نام کار: مطالعه بر روی ساختار Live CD و بررسی فایل‌سیستم‌های squashfs و unionfs	نگارش: ۱.۰	

فهرست مطالب

نام و نام خانوادگی: حامد جانزاده، وحیدکاظم‌پور	*تاریخ: 21/3/84*	
نام کار: مطالعه بر روی ساختار Live CD و بررسی فایل‌سیستم‌های squashfs و unionfs	نگارش: ۱.۰	

۱- مقدمه

یکی از بزرگترین مشکلات برای کسانی که می‌خواهند با سیستم عامل لینوکس آشنایی یابند، نصب و راه اندازی این سیستم عامل است که ممکن است برای مبتدیان غیر ممکن باشد. Live-CD ها این مشکل را به راحتی حل می‌کنند. Live-CD ها می‌توانند بدون نیاز به طی مراحل مورد نیاز برای نصب سیستم‌عامل و حتی با سخت‌افزار بسیار محدود به خصوص بدون نیاز به دیسک سخت نیز سیستم عامل لینوکس را در اختیار کاربران قرار دهند. این نوع سیستم‌عامل‌ها با وجود اینکه بر روی CD های غیر قابل نوشتنی اجرا می‌شوند امکان نوشتن بر روی شاخه‌های فایل سیستم را نیز به کاربر می‌دهند. به منظور راه‌اندازی سیستم عامل و همچنین برای فضایی که به منظور نوشتن در اختیار کاربر قرار داده می‌شود live-CD ها از فضای حافظه اصلی استفاده می‌کنند.

البته استفاده از live-CD ها را نمی‌توان جایگزین دایمی برای سیستم عامل‌های معمول دانست. زیرا این نوع استفاده از سیستم عامل نیز مشکلات خود را داراست. مهمترین آنها استفاده live-CD ها از حافظه اصلی برای تمام کارهایشان است که باعث می‌شود اولا حافظه زیادی مورد نیاز باشد و همچنین بعد از دوباره بوت کردن سیستم نیز تمام تغییراتی که در آن صورت گرفته بوده است از بین برود.

یکی از بهترین live-CD های موجود که امکانات زیادی را در اختیار کاربران قرار می‌دهد Slax است. ما نیز سعی داریم تا با ساختاری مشابه با ساختار Slax، ساخت یک live-CD را شرح می‌دهیم. ازمهمترین فواید Slax ساختار بر اساس ماژول آن است که امکان اضافه کردن و یا حذف کردن بسته‌های نرم‌افزاری به درون محتوای Live-CD و همچنین به جزییات kernel را راحت‌تر می‌نماید. در این مقاله، همچنین درباره ماژول‌ها و نحوه ساخت و استفاده کردن آنها و همچنین فایل سیستم‌هایی که ماژول‌ها بر اساس آنها ساخته می‌شوند نیز توضیحاتی ارایه خواهد شد.

۲- مراحل تولید یک Live-CD

در این گزارش در ابتدا مراحل بوت شدن یک سیستم‌عامل را شرح خواهیم داد و پس از آن توضیح خواهیم داد که چگونه می‌توان با داشتن یک سیستم عامل لینوکس که بر مبنای LFS-6.0 تولید شده باشد، یک Live-CD تولید کرد. در ادامه روش‌های مورد استفاده در Slax به منظور راحت‌تر شدن کار با Live-CD را بررسی خواهیم نمود و ساختارها و فایل سیستم‌های مورد استفاده در آن را شرح خواهیم داد.

۲-۱- چگونه یک Live-CD کار می‌کند

این عمل را می‌توان به سه مرحله تقسیم کرد:

- در مرحله اول، هسته لینوکس (vmlinuz) بارگذاری می‌شود، initrd.gz نیز درون یک فضای 4.4MB از حافظه اصلی استخراج شده و به عنوان فایل سیستم اصلی (root) نصب می‌شود.
- مرحله دوم با اجرا شدن دستورات درون linuxrc انجام خواهد شد که محتویات آن را دستورات شناخته شده برای لینوکس تشکیل می‌دهند و در هنگام ساخت Live-CD درون initrd.gz کپی می‌شوند. linuxrc فایل سیستم موقت (tmpfs) را درون mnt نصب کرده و تمام ماژول‌های اصلی سیستم را در شاخه اصلی فایل‌سیستم نصب می‌کند. در ادامه، ماژول‌های دلخواه نیز با دستورات

load= توسط kernel وارد فایل‌سیستم شده و سپس به درون شاخه/mnt عمل chroot انجام می‌شود.

- در مرحله سوم linuxrc فایل sbin/init را اجرا می‌کند که در واقع mnt/sbin/init است ولی به علت اینکه درون mnt، chrootشده است sbin/init خوانده می‌شود.
- از این مرحله به بعد تمام وظایف را سیستم عامل بر عهده دارد تا باقی کارها از جمله شناسایی سخت‌افزار و همچنین اجرا کردن برنامه‌های مورد نیاز را انجام دهد.

۲-۲- روش تولید یک CD بوت با استفاده از LFS-6.0

هدف از این مرحله تولید یک live-CD ساده است که با boot شدن کامپیوتر از روی آن یک linux prompt عرضه نماید. این دستورالعمل تنها برای معماری های منطبق با معماری کامپیوتری است که سیستم lfs رو آن پیاده سازی شده است.

در این قسمت سعی شده تا ساده ترین روش برای ساخت live-CD استفاده شود و برای این منظور از فایل سیستمهایی نظیر Squashfs و Unionfs استفاده نشده (البته در ادامه گزارش به این مفاهیم خواهیم پرداخت).

برای ساختن live-CD به این روش به دو سیستم مجزا نیازمندیم:

- یک سیستم کاری که پس از ساخته شدن live-CD برای write آن مورد استفاده قرار می‌گیرد .
- دیگری سیستمی که به طور خاص برای قرار گرفتن بر روی live-CD ساخته و آماده شده است.

اگر بخواهیم که CD ساخته شده فقط بر روی ماشین سازنده قابلیت boot شدن داشته باشد، آنگاه ایجاد یک کپی از سیستم LFS بر روی یک پارتیشن جدید کفایت می نماید.

به دو دلیل از دو سیستم مجزا برای ساخت live-CD استفاده می نماییم :

- اول اینکه اگر سیستم LFS رابرای کسب کارایی بیشتر بر روی دستگاهی خاص optimize کرده باشند، دیگر برای ساختن live-CD مناسب نمی باشد و برنامه ها بر روی دستگاه های غیر منطبق اجرا نخواهند شد.
- دیگر اینکه برای درست کردن CD نیازمندیم تا برخی پوشه ها را ایجاد کرده و فایل‌های بسیار مهمی را منتقل نماییم. در عمل اگر در این رویه با مشکلی مواجه شویم، استفاده از سیستم‌های مجزا این تضمین را فراهم می نماید که حداقل سیستم کاری موجود سالم خواهد ماند.

برای راحتی کار بهتر است که متغیر LIVECD را برای اشاره به mount point سیستمی که قرار است برروی CD استفاده شود، ایجاد نماییم

```
export LIVECD=/mnt/livecd
```

به همین طریق بهتر است متغیر CDDEV نیز ایجاد شود

```
export CDDEV=/dev/your-drive+partition
```

متغیر ISODIR را در مکتنی که مایل هستید تا image CD را در آن نگهداری نمایید، ایجاد کنید

```
export ISODIR=/where/you/have/space
```

حال که محیط لازم برای کار ایجاد شد، لازم است تا برخی از تنظیمات نهایی را برای سیستم live-CD انجام دهیم، که این تنظیمات

شامل اطمینان از تطبیق kernel با نیاز ما و احتمالا اضافه کردن برخی از برنامه های دیگر (بر حسب نیاز و یا تمایل شخصی) می باشد.
قبل از آنکه بتوانیم kernel مورد نیاز سیستم live-CD را تنظیم و نصب نماییم، باید به سیستم chroot کنیم، که برای این انجام مراحل زیر ضروری است.

mount کردن سیستم live-CD

```
mkdir -p $LIVECD
mount $CDDEV $LIVECD
```

mount کردن فایل سیستمهای مجازی (به همان طریق گفته شده در گزارش LFS-6.0)

```
mount -t proc proc $LIVECD/proc
mount -t sysfs sysfs $LIVECD/sys
```

و همچنین ایجاد fake mount های مربوطه

```
mount -f -t ramfs ramfs $LIVECD/dev
mount -f -t tmpfs tmpfs $LIVECD/dev/shm
mount -f -t devpts -o gid=4,mode=620 devpts $LIVECD/dev/pts
```

chroot به سیستم live-CD با استفاده از فرمانهای گفته شده در گزارش LFS-6.0

```
chroot $LIVECD /usr/bin/env -i \
HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin \
/bin/bash –login
```

mount کردن فایل سیستم ramfs و پر کردن dev

```
mount -n -t ramfs none /dev
/sbin/udevstart
```

ساختن دایرکتوری ها و لینک های ضروری که توسط udev ساخته نشده اند

```
ln -s /proc/self/fd /dev/fd
ln -s /proc/self/fd/0 /dev/stdin
ln -s /proc/self/fd/1 /dev/stdout
ln -s /proc/self/fd/2 /dev/stderr
```

```
ln -s /proc/kcore /dev/core
mkdir /dev/pts
mkdir /dev/shm
```

mount کردن فایل سیستمهای مجازی فراهم شده (برای kernel)

```
mount -t devpts -o gid=4,mode=620 none /dev/pts
mount -t tmpfs none /dev/shm
```

حال که در محیط chroot قرار گرفته ایم می توانیم kernel لازم برای CD را تنظیم کرده و بسازیم. ساختن هسته برای live-CD عمل پیچیده ای است. option هایی وجوبرای این منظور ابتدا باید پوشه و نقطه مرجعی برای ram disk ساخته شود.د دارند که باید درون هسته ساخته شوند و برخی از option ها نیز می توانند به صورت ماژول‌ها پیاده‌سازی شوند.
در مرحله بعدی مشاهده می‌شود که، در هنگام اجرای live-CD نیاز است تا برای برخی از فایل‌ها، امکان نوشتن وجود داشته باشد. پوشه fake/needwrite پوشه‌ای است که برای نگهداری اینگونه فایل‌ها در نظر گرفته شده است. به طور قطع این‌گونه فایل‌ها نمی‌توانند بر روی CD باقی بمانند، به همین دلیل در هنگام boot شدن CD این فایل‌ها به ram disk منتقل می‌شوند. قرار گرفتن این فایل‌ها بر روی ram disk به سیستم اجازه می دهد تا تغییرات ضروری را بر روی آنها اعمال نمایند.
برای این منظور ابتدا باید پوشه و نقطه مرجعی برای ram disk ساخته شود.

```
mkdir -p $LIVECD/fake/{needwrite,ramdisk}
```

سپس می توان پوشه هایی که نیاز است قابلیت نوشتن داشته باشند را روی آن منتقل کرد

```
cd $LIVECD/
mv dev/ etc/ home/ root/ tmp/ var/ fake/needwrite/
```

و در نهایت باید لینکهایی ساخته شود تا همه چیز همانند قبل در دسترس باشد

```
cd $LIVECD/
ln -s fake/needwrite/dev dev
ln -s fake/needwrite/var var
ln -s fake/needwrite/tmp tmp
ln -s fake/needwrite/root root
ln -s fake/needwrite/home home
ln -s fake/needwrite/etc etc
```

حال با اجرای دستور " ls –l " باید داشته باشیم

dev -> fake/needwrite/dev
etc -> fake/needwrite/etc
home -> fake/needwrite/home
root -> fake/needwrite/root
tmp -> fake/needwrite/tmp
var -> fake/needwrite/var

برای آنکه بتوانیم سرویس‌هایی را که نیازمند نوشتن بر روی /home/ /root /tmp /var/ /dev /etc هستند، اجرا نماییم باید با فراخوانی اسکریپتی از دایرکتوری /ram disk ، etc/rc.d/init.d را بر روی /fake/needwrite با دسترسی نوشتن mount کنیم.
اسکریپت زیر دو ram disk ایجاد می نماید. یک ram disk موقتی و یک ram disk که پوشه‌هایی که نیازمند دسترسی نوشتن هستند در آن قرار می گیرند.
این اسکریپت فایل‌ها را از روی CD بر روی ram disk موقتی کپی می‌نماید و سپس از آنجا به ram disk اصلی منتقل می کند.
می توان به جای دو ram disk از یک ram disk استفاده کرد ولی این عمل ممکن است مشکلات خاصی ایجاد نماید، به عنوان مثال initrd که در هنگام boot شدن سیستم بار می شود از اولین ram disk استفاده می نماید (/dev/ram0) و به همین دلیل هر تلاشی برای mount کردن dev/ram0/ به مکانی دیگر منجر به بروز خطای "device already mounted" خواهد شد، همچنین با umount کردن ram disk اطلاعات موجود بر روی آن از بین می رود و قابل دسترس نیست. لذا بهتر است از ram disk های متفاوت برای این منظور استفاده شود (بهتر است از وجود dev/ram2 , dev/ram1/ ،/dev/ram0 اطمینان حاصل نمایید).

```
cat > $LIVECD/etc/rc.d/init.d/create_ramdisk << "EOF"
#!/bin/sh

# SET UP SOME VARIABLES FOR DEVICES AND DIRECTORIES

dev_ram=/dev/ram1
dev_ram2=/dev/ram2
dir_ramdisk=/fake/ramdisk
dir_needwrite=/fake/needwrite

# SOURCE THE FUNCTIONS FILE
```

```
source /etc/rc.d/init.d/functions

case "$1" in
        start)

          # CREATE THE RAM DISK

          echo "Creating ext2fs on $dev_ram..."
         /sbin/mke2fs -m 0 -i 1024 -q $dev_ram > /dev/null 2>&1
         evaluate_retval
         sleep 1

        # MOUNT THE RAM DISK

        echo "Mounting ramdisk on $dir_ramdisk..."
        mount -n $dev_ram $dir_ramdisk -t ext2
        evaluate_retval
        sleep 1

        # COPY FILES TO THE RAM DISK

        echo "Copying files to ramdisk..."
        cp -a $dir_needwrite/* $dir_ramdisk > /dev/null 2>&1
        evaluate_retval
        sleep 1

        # CREATE SECOND RAMDISK

        echo "Creating second ramdisk"
        /sbin/mke2fs -m 0 -i 1024 -q $dev_ram2 > /dev/null 2>&1
        evaluate_retval
```

```
        sleep 1

        # MOUNT SECOND RAMDISK

        echo "Mounting second ram disk"
        mount -n $dev_ram2 $dir_needwrite -t ext2
        evaluate_retval
        sleep 1

        # COPY FILES TO THE SECOND RAMDISK

        echo "Copying files to the second ram disk"
        cp -a $dir_ramdisk/* $dir_needwrite
        evaluate_retval
        sleep 1

        # UNMOUNT THE FIRST RAMDISK

        echo "Unmounting and clearing first ram disk"
        umount -n $dir_ramdisk > /dev/null 2>&1
        blockdev --flushbufs /dev/ram1
        evaluate_retval
        sleep 1
        ;;
      *)
        echo "Usage: $0 {start}"
        exit 1
        ;;
esac

EOF
```

با استفاده از دستور زیر اسکریپت ایجاد شده در بالا را می توان اجرایی نمود :

```
chmod 0755 $LIVECD/etc/rc.d/init.d/create_ramdisk
```

حال لینک زیر را بسازید (توجه شود که s00 می تواند s11 نیز باشد به هر حال این سیستم lfs است و می توان ساز و کار ها را متناسب با نیازهای مشخصی تغییر داد)، نکته حائز اهمیت این است که باید اطمینان حاصل کرد که هیچ اسکریپتی که نیازمند اعمال تغییرات بر روی دایرکتوری های مشخص شده است، تا قبل از کپی این دایرکتوری ها بر روی ram disk ، اجرا نشود.

```
cd $LIVECD/etc/rc.d/rcsysinit.d
ln -s ../init.d/create_ramdisk S00create_ramdisk
```

در مرحله بعد باید boot loader و isolinux را نصب کرد که هر دوی آنها بر روی بسته syslinux موجودند http://freshmeat.net/projects/syslinux/
پس از دریافت بسته آن را در $LIVECD/usr/src قرار دهید و دستورات زیر را اجرا نمایید:

```
cd $LIVECD/usr/src

tar xzf syslinux-2.11.tar.gz
mkdir $LIVECD/isolinux
cp syslinux-2.11/isolinux.bin $LIVECD/isolinux

mv $LIVECD/boot/* $LIVECD/isolinux
cd $LIVECD/
rmdir boot
ln -s isolinux boot
```

برای boot loader باید فایل تنظیمات فراهم شود که دستورات زیر این کار را انجام می دهند

```
cat > $LIVECD/isolinux/isolinux.cfg << "EOF"
default livecd

label livecd
  kernel lfskernel
  append initrd=initrd.gz root=/dev/ram0 init=/linuxrc ramdisk_size=16384
EOF
```

برای آنکه پروسه boot شدن به نحو دلخواه ما انجام گیرد یک ram disk اولیه (initrd) می سازیم، ولی قبل از آن بهتر است محتوی فایل /etc/fstab را بر روی live-CD تغییر دهیم. بدین منظور تمامی اطلاعات غیر ضروری (به جز proc و devpts) را پاک می نماییم.
نگران mount شدن فایل سیستم root نباشید، این عمل توسط اسکریپت linuxrc در ram disk اولیه اجرا می شود.
بهتر است که لینک های زیر را پاک نمایید

```
rm $LIVECD/etc/rc.d/rc3.d/S20network
rm $LIVECD/etc/rc.d/rc0.d/K80network
rm $LIVECD/etc/rc.d/rc6.d/K80network
rm $LIVECD/etc/rc.d/rcsysinit.d/S40mountfs
rm $LIVECD/etc/rc.d/rcsysinit.d/S30checkfs
```

دستورات زیر initrd را می سازند:

```
dd if=/dev/zero of=$LIVECD/boot/initrd bs=1024 count=6144
mke2fs -m 0 -i 1024 -F $LIVECD/boot/initrd

mount -o loop $LIVECD/boot/initrd $LIVECD/mnt
cd $LIVECD/mnt
mkdir bin sbin lib dev proc mnt sys etc

cp -a $LIVECD/bin/{bash,mount,grep,umount,echo,ln,mkdir} bin/
cp -a $LIVECD/sbin/udev* sbin/
cp -a $(find $LIVECD -name "test" -type f) bin/
cp -a $(find $LIVECD -name "chroot" -type f) bin/
cp -a $(find $LIVECD -name "pivot_root" -type f) bin/
cp -H $LIVECD/lib/{libncurses.so.5,libdl.so.2,libc.so.6,ld-linux.so.2} lib/
        cp -H $LIVECD/lib/{libreadline.so.5.0,libhistory.so.5.0} lib/
cp -a $LIVECD/dev/{console,null,ram{0,1,2}} dev/
cp -a $LIVECD/etc/{udev,dev.d,hotplug.d} etc/

ln -s bash bin/sh
ln -s test bin/[
```

اولین برنامه ای که توسط kernel اجرا می شود linuxrc است که باید آن را بسازیم. این اسکریپت CD را در مکان صحیح تشخیص داده و آن را به عنوان فایل سیستم mount ، root می نماید و برنامه /sbin/init 3 را اجرا می کند.

```
cat > $LIVECD/mnt/linuxrc << "EOF"
#!/bin/sh

# ID is a file in root of the LFS boot CD, used to identify the CD.

ID="livecd"
TMP_MOUNT="/mnt"

PATH="/bin:/sbin:/usr/bin:/usr/sbin"

CHECK_TYPE="try_mount"

# MOUNT KERNEL FILESYSTEMS

# Create the proc directory if it does not exist

if [ ! -d "/proc/" ]; then
  mkdir /proc
fi

# Mount the proc filesystem

mount -n proc /proc -t proc

# If sysfs is listed as a valid filesystem type in /proc
# then mount it (if it doesnt then udev wont work
# and you wont have the devices you need)

if grep -q '[[:space:]]sysfs' /proc/filesystems; then
```

```
    if [ ! -d /sys/block ]; then
    mount -n sysfs /sys -t sysfs
    fi
fi

# Create some things that sysfs does not, and should not export for us.  Feel
# free to add devices to this list.

make_extra_nodes() {
        ln -s /proc/self/fd /dev/fd
        ln -s /proc/self/fd/0 /dev/stdin
        ln -s /proc/self/fd/1 /dev/stdout
        ln -s /proc/self/fd/2 /dev/stderr
        ln -s /proc/kcore /dev/core
        mkdir /dev/pts
        mkdir /dev/shm
}


if [ ! -x /sbin/hotplug ]; then
    echo /sbin/udev > /proc/sys/kernel/hotplug
fi

# Mount a temporary file system over /dev, so that any devices
# made or removed during this boot don't affect the next one.
# The reason we don't write to mtab is because we don't ever
# want /dev to be unavailable (such as by `umount -a').

mount -n ramfs /dev -t ramfs

/sbin/udevstart
```

```
make_extra_nodes
# Detecting the live CD is pretty complicated,
# but is a very logical process

#1. Search for cdrom devices and add them to CDROM_LIST

CDROM_LIST=""

# Search in proc tree for ide cdrom devices
# There used to be a section for devfs, but this was
# edited for udev. Actually we should probably not
# use /proc anymore, but use sysfs instead...
# Perhaps in the future;)

 # Check for ide channels.

 for ide_channel in /proc/ide/ide[0-9]
 do

  # If there are no ide channels found, then skip this

  if [ ! -d "$ide_channel" ]; then
   break
  fi

                # Try each ide device to see if we can find the cd-rom drive

  for ide_device in hda hdb hdc hdd hde hdf hdg hdh hdi hdj hdk hdl hdm hdn
  do
   device_media_file="$ide_channel/$ide_device/media"
```

```
    if [ -e "$device_media_file" ]; then
      grep -i "cdrom" $device_media_file > /dev/null 2>&1
      if [ $? -eq 0 ]; then
         CDROM_LIST="$CDROM_LIST /dev/$ide_device"
      fi
    fi
  done
done

# Check for scsi cds

for scsi_cdrom in /dev/scd[0-99]
do
  if [ -e "$scsi_cdrom" ]; then
    CDROM_LIST="$CDROM_LIST $scsi_cdrom"
  fi
done

#2. now we try to find the LFS boot CD (we use ID as identification)

LFS_CDROM_DEVICE=""

for cdrom_device in $CDROM_LIST
do
  if [ "$CHECK_TYPE" = "try_mount" ]; then
    mount -n -t iso9660 ${cdrom_device} $TMP_MOUNT
    # > /dev/null 2>&1
    media_found=$?
  fi
```

```
if [ $media_found -eq 0 ]; then

  echo -n "media found"
  if [ "$CHECK_TYPE" = "try_mount" ]; then
    [ -e "$TMP_MOUNT/$ID" ]
    media_lfs=$?
  fi
  if [ "$CHECK_TYPE" = "isoinfo" ]; then
   isoinfo -d -i $cdrom_device | grep -i "Volume id:" | grep "$ID" \
    > /dev/null 2>&1
    media_lfs=$?
    if [ $media_lfs -ne 0 ]; then
      isoinfo -V $cdrom_device | grep "$ID" > /dev/null 2>&1
      media_lfs=$?
    fi
  fi

  if [ "$CHECK_TYPE" = "try_mount" ]; then
    umount -n $cdrom_device > /dev/null 2>&1
  fi

  if [ $media_lfs -eq 0 ]; then
    echo ", LFS boot CD found. Ready!"
    LFS_CDROM_DEVICE="$cdrom_device"
    break;
  else
    echo ", not LFS boot CD."
  fi

else
  echo "no media "
```

```
  fi
done


#3. mount LFS CD as / (root fs)
if [ "$LFS_CDROM_DEVICE" = "" ]; then

  echo "No LFS boot CD found!!!"
  exit 1

else

  echo "Booting from $LFS_CDROM_DEVICE..."


# This is the magical part that makes a live CD live!
# The cd is mounted and pivot_root+chroot commands
# are used to start the system.
# If you really want to know what is going on here,
# You should read the chroot and pivot_root man pages
  mount -n -o ro -t iso9660 $LFS_CDROM_DEVICE $TMP_MOUNT
  cd $TMP_MOUNT
  pivot_root . mnt
  umount -n /mnt/proc >/dev/null 2>&1
  exec chroot . sh -c 'umount -n /mnt >/dev/null 2>&1; exec -a init.new /sbin/init 3' <dev/console >dev/console 2>&1

fi

EOF
```

در ادامه دستورات زیر را اجرایی نمایید

```
    chmod 0755 $LIVECD/mnt/linuxrc
```

```
cd $LIVECD/
umount $LIVECD/mnt
gzip $LIVECD/boot/initrd
```

در نهایت برای write کردن live-CD اعمال زیر را باید انجام دهیم.
ابتدا اندازه شاخه lfs را چک می نماییم و تمامی شاخه های اضافی را حذف می نماییم (مانند /usr/src/*).
از آنجایی که linuxrc باید بتواند CD ساخته شده را شناسایی کند، باید فایلی با عنوان livecd ساخته شود.

```
touch $LIVECD/livecd
```

حال سیستم lfs را می توان بر روی write کرد.

```
/cd $LIVECD
mkisofs -R -l -L -D -b isolinux/isolinux.bin -o $ISODIR/livecd_image.iso -c isolinux/boot.cat -no-emul-boot -boot-load-
&& size 4 -boot-info-table -V "livecd" $LIVECD
cdrecord -v -eject dev=/dev/hdc blank=fast $ISODIR/livecd_image.iso
```

۲-۳- روشهای اضافه کردن و حذف کردن نرم‌افزارها به مجموعه Live-CD

یکی از این معایب مصرف حافظه بیش از حد معمول آن نسبت به لینوکس‌های معمولی است. علت نیز بار شدن تمام فایل سیستم‌هساختار Live-CD می‌تواند بسیار ساده باشد، به این معنی که تمام نرم‌افزارهای مورد نیاز به همراه لینوکس در شاخه اصلی نصب شوند و در نهایت با روشی که در بخش قبل توضیح داده شد این فایل‌ها بر روی CD منتقل شده و Live-CD را تولید کنند. ولی این روش دارای معایبی است. ای قابل نوشتن آن بر روی حافظه اصلی است و این کار بهره وری در استفاده بهینه از حافظه را کاهش خواهد داد. مشکل دیگر این روش در حذف کردن و اضافه کردن نرم‌افزارها به یا از درون فایل سیستم است. اضافه یا حذف کردن یک برنامه از لینوکس در این روش نیاز به نوشتن دوباره Live-CD درون دیسک دارد.

یکی از بهترین روش‌هایی که توسط Slax برای حل مشکلات نامبرده استفاده می‌شود، استفاده از ماژول‌ها است. در Slax هر کدام از نرم‌افزارها درون شاخه‌ای خاص نصب می‌شوند ومحتویات آن شاخه که شامل تمام فایل‌های مورد نیاز نرم‌افزار به همراه شاخه‌هایشان است توسط یک نوع فایل سیستم به عنوان مثال squashfs فشرده‌سازی شده و تبدیل به یک ماژول (یک فایل .mo) می‌شود. با mount کردن هر کدام از این ماژول‌ها به وسیله ساختاری همچون unionfs می‌توان محتویات این ماژول‌ها را به محتویات فایل سیستم اصلی اضافه کرد، به گونه‌ای که گویا فایل‌های این نرم‌افزار از ابتدا درون فایل سیستم نصب شده بوده‌اند. حذف کردن یک نرم افزار نیز به راحتی unmount کردن ماژول آن انجام می‌شود. در نتیجه اضافه یا حذف کردن یک بسته نرم‌افزاری به مجموعه Live-CD بسیار راحت‌تر شده و همچنین با mount و unmount کردن ماژول‌ها بر حسب نیاز می‌توان در استفاده از حافظه نیز صرفه‌جویی نمود. در ادامه روش تولید و استفاده از

ماژولها را توضیح خواهیم داد.

۲-۳-۱- ماژول‌ها

ماژول‌ها، فایل‌هایی هستند که با ساختار یک نوع فایل سیستم مشخص ساخته شده‌اند. ماژول در واقع محتوی چیزی جز تعدادی فایل و شاخه که همراه با هم درون آن فشرده شده‌اند نمی‌باشد. ماژول‌هایی که توسط Slax مورد استفاده قرار می‌گیرند با استفاده از فایل سیستم squashfs ساخته می‌شوند و با پسوند .mo شناخته می‌شوند.

۲-۳-۲- چگونه یک ماژول ساخته می‌شود

روش‌های زیادی برای تولید ماژول‌ها وجود دارند که ما در اینجا یکی از آنها را شرح خواهیم داد. تمام دستوراتی که ما در اینجا از آنها استفاده می‌کنیم قابل استفاده در Slax و همچنین Linux مورد استفاده شما می‌باشند ولی در صورت نیاز می‌توانید آنها را تهیه و نصب نمایید.
با استفاده از دستور زیر می‌توان یک بسته TGZ را به یک ماژول تبدیل نمود:

```
$ tgz2mo application.tgz application.mo
```

اگر بخواهید قبل از تبدیل بسته به ماژول محتویات آنرا تغییر دهید می‌توانید از installpkg استفاده کنید:

```
$ installpkg --root /tmp/package application.tgz
```

این دستور بسته TGZ را درون شاخه متفاوتی (/tmp/package) نصب می‌نماید. بعد از انجام تغییرات، شاخه حاصل را با دستور زیر به ماژول تبدیل کنید:

```
$ dir2mo /tmp/package module.mo
```

تمام ماژول‌ها با دستور mksquashfs ساخته می‌شوند که جزء دستورات شناخته شده برای لینوکس است. شما همچنین می‌توانید ابزار squashfs را تهیه نموده و آنرا نصب کنید ولی kernel مورد استفاده شما برای اجرای دستورات dir2mo و tgz2mo نیازی به squashfs ندارد و تنها فایل باینری mksquashfs لازم است.

۲-۳-۳- چگونه می‌توان محتوای یک ماژول موجود را تغییر داد

برای دستکاری ماژول‌ها لازم است kernel شما فایل سیستم squashfs را پشتیبانی کند. دستور زیر محتوای یک ماژول را درون شاخه / tmp باز می‌کند:

```
$ mo2dir module.mo /tmp/directory
```

شما می‌توانید محتویات ماژول را از درون شاخه /tmp/directory به هر شکلی که بخواهید تغییر دهید و در نهایت شاخه را به ساختار ماژولها برگردانید.

```
$ dir2mo /tmp/directory module.mo
```

اگر می‌خواهید محتویات درون ماژول را نیز مشاهده کنید، می‌توانید آنرا بر روی یک شاخه نصب کنید:

```
$ mount -t squashfs -o loop \
> /path/module.mo \
> /mnt/mountpoint
```

۲-۳-۴- چگونه می‌توان از ماژول‌ها استفاده کرد

راه‌های زیادی برای استفاده از یک ماژول وجود دارد. می‌توان آنرا از ابتدای بارگزاری شدن سیستم درون فایل سیستم اصلی نصب کرد یا می‌توان آنرا در اختیار کاربر قرار داد تا هر وقت لازم دید نصبش نماید. روش اول همیشگی خواهند می‌شود زیرا در طول حیات سیستم باقی می‌ماند.
ماژول‌ها با استفاده از فایل سیستمی همچون unionfs درون شاخه اصلی اضافه می‌شوند. در ادامه درباره unionfs توضیح داده خواهد شد. با استفاده از ابزارهای Slax نیز می‌توان این کار را انجام داد. Slax با استفاده از ابزار uselivemod یک ماژول را به شکل زیر نصب می‌نماید:

```
$ uselivemod /path/module.mo
```

دراین حالت ماژول بلافاصله درون سیستم اضافه می‌شود، در نتیجه به نظر خواهد آمد که نرم‌افزار نصب شده است.
در ادامه در رابطه با فایل سیستم‌های squashfs و unionfs توضیح داده خواهد شد که ماژولها بر اساس آنها ساخته شده و درون فایل سیستم نصب می‌شوند.

۳- فایل‌سیستم

در این بخش فایل‌سیستم‌های squashfs و unionfs شرح داده خواهد شد.

۳-۱- Squashfs

تا این مرحله با نحوه ساخت یک live-CD ، اسکریپت‌های استفاده شده و فایل‌های منتقل شده بر روی CD آشنا شدیم. همانگونه که قبلا نیز اشاره شد در دستورالعمل‌ها و توضیحات بیان شده از فایل سیستمهای squashfs و unionfs برای ساده کردن کار استفاده نشده بود و در نهایت live-CD ارائه شده نیز بسیار ساده بوده و تنها یک command line linux را برای کاربر فراهم می کرد. در ادامه بحث نیز اشاره شد برای افزایش قابلیت‌ها نیازمند نصب برنامه های بیشتری هستیم که در این رابطه به کاربرد module ها نیز اشاره شد.

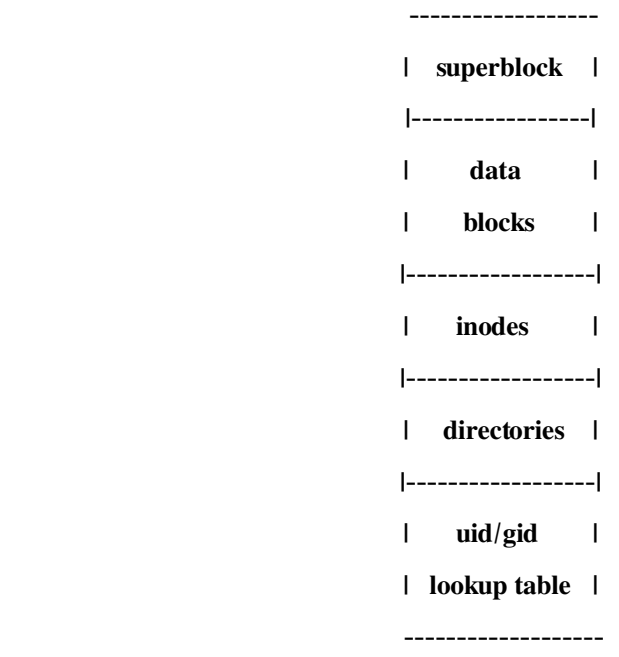
برای استفاده بهینه از فضای محدودِ CD بکاربردن فایل سیستم‌هایی نظیرِ cramfs وِ squashfs ِکه عمل فشرده‌سازی را بر روی یک دایرکتوری و با یک پارتیشن انجام می دهند، ضروری به نظر می رسد. در مقایسه‌های کارایی و انعطاف پذیری مابین این فایل سیستم‌ها، مشخص شده که squashfs از قابلیت بالایی برخوردار است. لذا در ادامه به بررسی برخی مزایای این فایل سیستم و نحوه نصب و استفاده از آن می پردازیم.

فایل سیستم squashfs یک فایل سیستم فشرده شده و فقط خواندنی برای linux است که برای استفاده در سیستم‌های با اندازه بسیار کوچک، سیستمهای embedded و یا هر جا که نیازمند استفاده از فایل سیستم‌های فشرده شده هستیم، تهیه شده است.

در اینگونه سیستم‌ها هر بایت از فضای ذخیره‌سازی دارای ارزش می باشد، لذا فشرده‌سازی در هر جا که مقدور باشد مورد استفاده قرار می‌گیرد. علاوه بر این از فایل سیستم‌های فشرده شده برای مقاصد ذخیره‌سازی و آرشیو نیز به صورت گسترده‌ای استفاده می‌شود.

squashfs اجازه می‌دهد تا تمامی فایل‌سیستم و یا فقط یک دایرکتوری را فشرده کرده و آن را بر روی یک فایل یا پارتیشن نوشته و ذخیره کنید و سپس با mount کردن آن پارتیشن‌ها و یا استفاده از loop back device ها برای فایل‌ها، اطلاعات فشرده شده را بازیابی نمایید. ساختار و طراحی modular ای فایل سیستم، انعطاف پذیری و کارایی بالایی را برای آن فراهم آورده است (برخی از این مقایسه ها در مستند PERFORMANCE.README در squashfs package آورده شده است).

فایل سیستم squashfs دارای پنج قسمت است که به صورت همتراز در کنار یکدیگر قرار گرفته اند (شکل ۳-۱).



شکل ۳-۱- ساختار فایل‌سیستم squashfs

بلوک‌های اطلاعات که فشرده شده‌اند، به صورت فایل‌هایی که از دایرکتوری مرجع خوانده می شوند، در فایل سیستم نوشته شده و برای یافتن تکرار تست می شوند. بعد از اینکه تمامی اطلاعات نوشته شدـ inode ـهای تکمیل شده، پوشه‌ها و جداول جستجویـ userid ـو groupid نوشته می‌شوند. متادیتاها (inode ها و پوشه ها) در بلوکهای kbyte 8 فشرده می شوند. هر بلوک فشرده شده دارای پیشوندی به طول byte 2 است که بیت اول آن مشخص می نماید که آیا بلوک فشرده شده است یا خیر. inode ها در این بلوک‌های متادیتا قرار دارند ولی توسط مرزهای این بلوک‌ها تراز نمشوند به همین دلیل inode ها بلوک‌های فشرده شده را همپوشانی می نمایند. هر inode توسط دو فیلد مشخص می شود (آدرس شروع بلوک فشرده شده و آفست آن در بلوک باز شده).

فایل‌سیستم squashfs تمامی اطلاعات، inode ها و پوشه‌ها را فشرده می‌نماید و همچنین 32 bit اطلاعت در رابطه با usreid و groupid و تاریخ ساخته شدن فایل، ذخیره می‌نماید. این فایل سیستم می‌تواند از block size هایی با حداکثر اندازه 32 kb در نگارش x.1 و 64 kb در نگارش x.2 استفاده نماید که این block size ها سبب افزایش نرخ فشرده‌سازی نسبت به block size های معمولی (4 kb) می‌شود.

در squashfs 2.x امکان قرار دادن فایل‌هایی با اندازه کمتر از یک block در، یک block واحد (fragment blocks) وجود دارد، که نتیجه آن افزایش نرخ فشرده‌سازی است.

در این فایل سیستم فایل‌های تکراری شناسایی و حذف می شوند و هر دو نوع معماری little endian و big endian پشتیبانی می شود. همچنین این فایل‌سیستم دارای این قابلیت است که فایل‌سیستم‌های ساخته شده توسط ماشین‌هایی با byte code های متفاوت را mount نماید.

squashfs به عنوان یک patch برای هسته linux (که امکان خواندن و پشتیبانی از squashfs را برای kernel فراهم می نماید) گسترش

یافته است و دارای ابزاری به نام mksquashfs است، که امکان ساخت این فایل سیستم را فراهم می‌نماید. برای خواندن squashfs درست مانند هر فایل سیستم دیگری، هسته باید از آن پشتیبانی نماید. برای این منظور patch هایی برای kernel نگارش های مختلف ایجاد شده که در شاخه linux-2.x.y در بسته squashfs وجود دارد (لازم به تذکر است که برای patch کردن هسته معمولا نمی توان از سورس‌های تهیه شده توسط شرکت های مختلف استفاده نمود زیرا آنها خود patch شده هستند و patch آنها احتمالا عمل نخواهد کرد). فرض بر این است که سورس‌ها در شاخه /usr/src نگهداری می شوند. حال patch های مربوط به هسته را کپی نمایید

```
cd /usr/src/squashfs
cp linux-2.x.y/squashfs-patch /usr/src/linux
```

و سپس با رفتن به دایرکتوری هسته آن را patch می نماییم

```
cd /usr/src/linux
patch -p1 < squashfs-patch
```

پس از patch کردن هسته، لازم است تا هسته دوباره تنظیم و کامپایل شود

```
make distclean
make mrproper
make menuconfig
```

در تنظیم هسته به چند نکته باید توجه شود

- در شاخه فایل سیستم، زیر شاخه Miscellaneous File Systems گزینه Squashfs filesystem باید فعال شود.
- اگر مایل به استفاده از Squashfs initial ram disk هستید گزینه مربوط به آن (Initial RAM disk Support) را در شاخه device driversو زیر شاخه Block devices فعال نمایید.
- اگر می خواهید فایل سیستم squashfs را بوسیله loop back device ها mountنمایید گزینه loopback device support را نیز در این شاخه فعال نمایید.

(این دستورالعمل برای هسته های نگارش x.2.6 می باشد و برای نگارش های پایین تر به مقاله HOWTO Squashfs واقع در http://artemio.net/projects/linuxdoc/squashfs مراجعه نمایید).

پس از کامپایل هسته لازم است تا image kernel را در شاخه /boot کپی نمایید و پس از آن ماژول های هسته را (در صورت وجود) نصب کرده و boot loader را دوباره تنظیم نمود تا هسته جدید را بشناسد.

```
cp arch/i386/boot/bzImage /boot/bzImage-sqsh
make modules_install
```

```
cat /proc/filesystems
```

در مرحله بعد باید ابزار mksquashfs را نصب کرد تا به کمک آن بتوان فایل سیستم squashfs را ساخت.

```
cd /usr/src/squashfs/squashfs-tools
make
cp mksquashfs /usr/sbin
```

حال می توان به راحتی از squashfs برای مقاصد مختلف استفاده نمود. در اینجا به نحوه ساخت و استفاده از این فایل سیستم می پردازیم. همانگونه که قبلا نیز اشاره شد استفاده از این فایل سیستم بسیار ساده و با حداقل دستورات امکان پذیر است برای نمونه اگر بخواهیم چند دایرکتوری را توسط این فایل سیستم فشرده نماییم باید دستورات زیر را اجرا کنیم

```
mksquashfs /some/dir dir.sqsh
```

که پس از اجرای این دستور image ایجاد شده را تحت نام dir.sqsh موجود است، می توان mount کرده و اطلاعات آن را بازیابی نمود.

```
mkdir /mnt/dir
mount dir.sqsh /mnt/dir -t squashfs -o loop
```

در کاربرد فایل سیستم squashfs برای سیستمهای linux بر روی فضاهای کوچک (مانند live-CD)، در راستای مراحل تولید این گونه سیستم ها باید کارکردهای زیر را در مواقع مقتضی به کار برد.

- در هنگام تهیه هسته برای سیستم مطمئن شوید که هسته پشتیبانی لازم را از فایل سیستم squashfs به عمل می آورد.
- از mksquashfs برای تولید ram disk اولیه و یا سایر فایل سیستمها استفاده نمایید
- ـ در اسکریپت های startup ویا فایل /etc/fstab حتما نوع squashfs را تنظیم نمایید تا این فایل سیستم قابل mount کردن باشد.

نکته باقی مانده این است که، فایل سیستم squashfs یک فایل سیستم فقط خواندنی است و همانطور که در توضیح ساخت یک live-CD ساده بیان شد در هنگام boot برخی از اسکریپت‌ها نیاز به دسترسی نوشتن بر روی برخی پوشه ها دارند. لذا از فایل سیستم unionfs استفاده می شود که اجازه می دهد که تمامی image فایل‌ها حالت pseudo writeable داشته باشند. توضیحات بیشتر در این زمینه در ادامه آورده شده اند.

Unionfs -۲-۳

Unionfs، که در سال ۲۰۰۴ در دانشگاه stoony brook تولید شده، نوعی از فایل سیستم است که قابلیت ترکیب چند شاخه یا فایل سیستم در یک فایل سیستم دیگر، به شکلی که محتوای فایل سیستم‌ها و شاخه‌های اصلی به صورت جداگانه نگهداری شوند، را در اختیار

می‌گذارد. با استفاده از این روش می‌توان انواع مختلفی از شاخه‌ها را با انواع متفاوتی از دسترسی‌های خواندن و نوشتن در یک فایل سیستم جدید تولید کرد. همچنین می‌توان بر روی این شاخه‌ها پاک کردن و ایجاد کردن‌های مجازی اعمال نمود. از Unionfs در موارد زیادی می‌توان بهره برد. به عنوان مثال برای ترکیب شاخه‌های home از فایل سیستم‌های متفاوت و بر روی دیسک پارتیشن‌های مختلف، یا به منظور ترکیب چند CD برای ایجاد یک آلبوم یکتا از تصاویر می‌توان آنرا به کار برد. علاوه بر این با استفاده از قابلیت "کپی در هنگام نوشتن" در این فایل سیستم می‌توان دو شاخه جداگانه یکی با دسترسی فقط خواندن و دیگری با دسترسی خواندن و نوشتن را ترکیب نمود و با کپی گرفتن از نسخه جدید فایل‌های فقط خواندنی که کاربر عمل نوشتن را بر روی آنها انجام میدهد در یکی از شاخه‌های با دسترسی نوشتن، این تصور را برای کاربر ایجاد کرد که درون شاخه‌های غیر قابل نوشتنی می‌تواند بنویسد.
Unionfs ـبه عنوان مهمترین بخش ازـ Slax ـبه حساب می‌آید و باعث می‌شودـ Slax ـهمانند سیستم عامل‌هایـ Linux ـمعمول با قابلیت دسترسی کامل برای نوشتن در شاخه root عمل نماید.

۳-۲-۱- نصب Unionfs

برای آنکه بتوانید از Unionfs استفاده کنید در وحله اول لازم است که kernel ای را نصب نمایید که Unionfs را پشتیبانی کند. Unionfs به عنوان یکی از ماژولهای kernel در نسخه‌های 2.4.20 تا 2.6.9 و بالاتر وجود دارد ولی در صورت نیاز می‌توان بسته‌های آنرا تهیه نموده و به صورت دستی آنرا نصب نمود. برای نصب آن باید ابتدا بسته فشرده شده آنرا باز کرد:

```
$ tar –xzf unionfs-x.y.z.tar.gz
```

سپس وارد شاخه‌ای که بسته در آن باز شده شوید و دستورات زیر را اجرا کنید:

```
$ make
$ make install
$ depmod –a
```

۳-۲-۲- استفاده از Unionfs

در ساختار Unionfs هر شاخه دارای یک اولویت است. شاخه با اولویت بالاتر، شاخه با اولویت پایینتر را بازنویسی می‌کند. اگر در دو شاخه ترکیب شونده دو زیر شاخه هم اسم وجود داشته باشد، زیر شاخه ایجاد شده در ساختار Unionfs شامل محتویات هر دو شاخه اولیه خواهد بود. Unionfs به طور خودکار شاخه‌های تکراری را حذف می‌نماید. در نتیجه دچار سر درگمی به علت دیدن شاخه‌های هم‌نام نخواهند شد. اگر نام فایلی نیز در هر دو شاخه تکرار شده بود، محتوا و جزییات فایل ایجاد شده در ساختار Unionfs هانند فایل موجود در شاخه با اولویت بالاتر خواهد بود و فایل موجود د ر شاخه با اولویت پایینتر نادیده گرفته می‌شود.
به عنوان مثال فرض کنید می‌خواهیم دو شاخه با نام‌های /Fruits و /Vegetables را با ساختار Unionfs ترکیب کنیم. درون شاخه /Fruits فایلهای Apple و Tomato وجود دارند و درون شاخه /Vegetables نیز فایلهای Carrots و Tomato. در فایل /Fruits/Tomato نوشته شده "

I_am_botanically_a_fruit." و در فایل /Vegetables/Tomato نیز نوشته شده "I_am_horticulturally_a_vegetable.".
برای یکپارچه کردن این دو شاخه، Unionfs را به شکل زیر mount خواهیم کرد:

```
$ mount –t unionfs –o \
> dirs=/Fruits:/Vegetables \
> none /mnt/healthy
```

در این مثال dirs= نشان می‌دهد که کدام شاخه‌ها باید با یکدیگر یکپارچه شوند و ترتیب بیان کردن آنها نیز اولویت آنها را مشخص می‌نماید. بعد از اجرای این دستور، محتوای شاخه /mnt/healthy شامل سه فایل Apple و Carrots و Tomato خواهد بود و از آنجا که ما / Fruits را با اولویت بالاتری نسبت به /Vegetables مشخص کرده‌ایم، محتوای فایل /mnt/healthy/Tomato "I_am_botanically_a_fruit." خواهد بود.

۳-۲-۳- روش "کپی در هنگام نوشتن" در Unionfs

در مثال قبل شاخه‌های اولیه از نوع فقط خواندنی بودند در نتیجه فایل سیستم ساخته شده توسط Unionfs نیز از نوع فقط خواندنی است. Unionfs همچنین می‌تواند فایل سیستم‌هایی از ترکیب شاخه‌های مختلف با انواع مختلفی از دسترسی‌های خواندن و نوشتن نیز تولید کند. در این حالت ساختار union به طور کلی قابل خواندن و نوشتن است و برای نوشتن از روش کپی در هنگام نوشتن استفاده می‌شود. در این روش نوشتن بر روی فایل‌های فقط خواندنی قابل انجام است ولی نسخه تغییریافته در شاخه با قابلیت نوشتن ایجاد می‌گردد. این کار را می‌توان بر روی یک CD-ROM انجام داد اگر CD-ROM در /mnt/cdrom مانت شده و یک شاخه خالی نیز در /tmp/cdpatch وجود داشته باشد. union می‌تواند به شکل زیر ساخته شود:

```
$ mount –t unionfs –o \
> dirs=/tmp/cdpatch:/mnt/cdrom \
> none  /mnt/patched-cdrom
```

هنگامی که درون شاخه /mnt/patched-cdrom را مشاهده می‌کنید به نظر می‌آید که می‌توان بر روی محتویات CD-ROM تغییراتی ایجاد کرد و چیزهایی نوشت ولی در واقع تمام تغییرات در شاخه /tmp/cdpatch نوشته می‌شود. زمانی که درخواست نوشتن بر روی یک فایل فقط خواندنی صادر می‌شود در هنگام ذخیره کردن تغییرات، فایل دیگری در یک شاخه با اولویت بالاتر که دسترسی نوشتن بر روی آن وجود داشته باشد ایجاد می‌شود. در صورتی که لازم باشد Unionfs شاخه‌های پدر را به صورت خودکار ایجاد می‌کند.
در برخی مواقع union از ترکیب شاخه‌هایی تشکیل می‌شود که دارای قابلیت نوشتن هستند ولی در ساختار Unionfs بنابه دلایلی

می‌خواهیم محتویات این شاخه‌ها دچار تغییرات نشوند. در این صورت می‌توان در هنگام ساختن union این شاخه‌ها را توسط علامت ro= فقط خواندنی فرض کردتا در آنها نوشتن صورت نگیرد. به عنوان مثال فرض کنید می‌خواهیم شاخه usr/src/Linux/ فقط خواندنی فرض شودو تغییراتی که در آن صورت خواهد گرفت در شاخه خالی home/cpw/Linux/ ذخیره شود. به این شکل می‌توان union ای با این مشخصات برای آن ایجاد کرد:

```
$ mount −t unionfs −o \
> dirs=/home/cpw/Linux:/usr/src/Linux=ro \
> none /home/cpw/Linux-src
```

همچنین می‌توان از همان شاخه home/cpw/Linux/ برای مقصد mount استفاده کرد:

```
$ mount −t unionfs −o \
> dirs=/home/cpw/Linux:/usr/src/Linux=ro \
> none /home/cpw/Linux
```

۳-۲-۴- استفاده از unionctl

unionctl یکی از ابزارهایی است که در هنگام کامپایل کردن بسته unionfs در مسیر usr/local/sbin/ نصب می‌شود. unionctl به منظور مدیریت کردن unionهای موجود از طریق list کردن، modify ، addو delete در آنها انجام می‌شود. در زیر تعدادی از نمونه‌های استفاده از unionctl را مشاهده می‌کنید.
لیست کردن شاخه‌های درون یک union بادستور زیر انجام می‌شود:

```
$ unionctl /mnt/union --list
```

اضافه کردن یک شاخه جدید به union موجود توسط دستور زیر انجام می‌شود:

```
$ unionctl /mnt/union --add \
> --[after یا before] /mnt/some/branch \
> --mode [ro یا rw] \
> /mnt/new/branch
```

به منظور حذف یک شاخه از یک union موجود نیز می‌توان روشهای زیر را مورد استفاده قرار داد:

- روش *WHITEOUT* : در این روش به جای پاک کردن تمام فایل‌های هم نام آنها را تبدیل به فایل .wh می‌کند و در نتیجه فایل‌ها قابل دسترس خواهند بود. این روش در حالت پیش‌فرض استفاده می‌شود.
- روش *DELETE_ALL* : تمام نسخه‌های موجود از فایل را از تمام شاخه‌ها با اولویت‌های متفاوت می‌کند.
- روش *DELETE_FIRST* : با استفاده از این روش تنها فایل با اولویت بالاتر پاک می‌شود و فایلهای با اولویت پایین‌تر قابل مشاهده خواهند بود.

روش پاک کردن را باید در هنگام ساختن union با استفاده از دستوری مشابه دستور زیر مشخص کرد:

```
$ mount −t unionfs −o \
> dirs=/home/cpw/Linux:/usr/src/Linux=ro \
> delete=first none /home/cpw/Linux
```