دانشگاه آزاد اسلامی

واحد تهران شمال

دانشکده فنی و مهندسی

گروه مهندسی کامپیوتر

پروژه کارشناسی

گرایش نرم‌افزار

عنوان:

طراحی و پیاده‌سازی یک سیستم‌عامل آزاد برای معماری x86

استاد راهنما:

مهندس علی رضایی

نگارش:

دانیال بهزادی

زمان:

نیمسال دوم ۹۱-۱۳۹۰

تقدیم‌نامه

تقدیم به «ریچارد استالمن»[1] و «جیمی ولز»[2] که با تلاش‌ها و از خودگذشتی‌هایشان آزادی را در عصر دیجتال دوباره معنا بخشیدند.

[1] Richard Matthew Stallman

[2] Jimmy Donal Wales

تقدیر و تشکر

با تشکر از پدر و مادرم که من را تحمل کردند

فهرست مطالب

چکیده

در این پایان‌نامه ابتدا با اجزای مختلف یک سیستم‌عامل آشنا شده و سپس با معرفی پیاده‌سازی‌های آزاد آن‌ها که اکثراً از پروژه‌ی گنو[۳] انتخاب شده‌اند، چگونگی قرار گرفتن این اجزا در کنار یک‌دیگر برای داشتن یک سیستم‌عامل کارا معرفی می‌شود. سپس اجزای مختلف برای قرار گرفتن در جایگاه‌های صحیح خود بسته به نیاز، با پیکربندی‌های متفاوت کامپایل شده و در پیکره‌ی سامانه قرار می‌گیرند تا درنهایت به طرح دل‌خواه بدل گردند. در انتها روشی بیان می‌گردد که به وسیله‌ی آن بتوان سیستم‌عامل ایجاد شده را مستقیماً از روی دیسک بالا آورده و اجرا کرد.

[۳] GNU: GNU's Not Unix

مقدمه

از زمانی که نخستین رایانه‌ها ساخته شدند، همواره برای استفاده‌ی بهتر از منابع، نیاز به نرم‌افزاری سیستمی بود که مدیریت نرم‌افزار کاربردی را برعهده بگیرند. به چنین نرم‌افزاری سیستم‌عامل[۴] گفته می‌شد. در طول سالیان شرکت‌های مختلفی دست به تولید سیستم‌عامل زدند که نتیجه‌ی آن تولید ده‌ها گونه‌ی مختلف از سیستم‌های عامل بود. با همه‌ی این اوصاف پس از چندی یک سیستم‌عامل به نام یونیکس[۵] توانست با قدرت بالا و راحتی استفاده‌ی خود دیگر رقبا را از صحنه‌ی رقابت حذف کرده و به استاندارد غیررسمی سیسستم‌های عامل بدل شود. با این حال یونیکس یک مشکل بزرگ داشت؛ کد منبع آن انحصاری و در اختیار شرکت تولید کننده‌ی آن بود. این امر از نظر «ریچارد متیو استالمن» که دغدغه‌ی زیرپا گذاشته شدن آزادی‌های انسانی در عصر دیجیتال را داشت، غیرقابل بخشش بود.

وی که به تازگی از کار خود در یک شرکت نرم‌افزاری استعفا داده بود تا بنیاد نرم‌افزار آزاد[۶] را تأسیس کند، به خوبی می‌دانست که داشتن نرم‌افزار آزاد در یک بستر انحصاری اثر چندانی ندارد. به همین دلیل دست‌به‌کار شد تا یک سیستم‌عامل آزاد تولید نماید. او این سیستم‌عامل را که قرار بود شبیه یونیکس باشد، امّا برخلاف یونیکس آزاد بود را GNU نامید که سرنام عبارت GNU's Not Unix است به معنای «گنو یونیکس نیست».

تیم توسعه‌دهنده‌ی گنو سال‌ها بر روی این سیستم‌عامل کار کرده و اکثر بخش‌هایی که یک

---

[۴] OS:Opertaing System

[۵] Unix

[۶] FSF: Free Software Foundation

سیستم‌عامل مدرن به آن نیاز دارد را پیاده‌سازی کردند. از کامپایلر زبان‌های مختلف برنامه‌نویسی[۷] گرفته تا چندین بازی سرگرم‌کننده. با این حال هسته‌ی این سیستم‌عامل[۸] که قرار بود نخستین هسته برمبنای معماری تازه مطرح شده‌ی ریزهسته[۹] باشد، به دلیل پیچیدگی‌های زیاد این معماری به زمان خیلی بیش‌تری برای پیاده سازی نیاز داشت.

در همان سال‌ها دانشجویی فنلاندی به نام «لینوس توروالدز»[۱۰] برای سرگرمی یک هسته‌ی یک‌پارچه[۱۱] ساخت و آن را به صورت آزاد روی اینترنت قرار داد. این هسته که بسیار خام بود توسّط توسعه‌دهندگان گنو مورد استقبال واقع شد و تصمیم بر این شد که تا زمان آماده شدن نسخه‌ی نهایی هِرد، به صورت موقّت از این هسته که لینوکس[۱۲] نام گرفت استفاده شود، پس اعضای تیم گنو شروع به توسعه‌ی این هسته‌ی جوان کردند. برای آن که سیستم‌عامل گنو با هسته‌ی لینوکس که اکنون می‌شد از آن استفاده کرد با سیستم‌عامل اصلی گنو اشتباه گرفته نشود، نام «گنو/لینوکس» بدان اطلاق شد.

با گذشت سالیان به دلیل معماری ساده‌تر لینوکس توسعه‌ی آن روند سریع‌تری به خود گرفت و گنو/لینوکس کارآیی‌ای معادل و بیش‌تر از نمونه‌های انحصاری پیدا کرد. از این رو بسیاری به استفاده از آن روی آوردند. با این‌حال هسته‌ی هِرد هم‌چنان در حال توسعه است و نسخه‌های پیش‌نمایش آن حکایت از کارآیی بسیار بالاتر آن نسبت به لینوکس دارند.

سیستم‌عاملی که در این پایان‌نامه به عنوان هدف درنظر گرفته شده است همان سیستم‌عامل گنو با هسته‌ی لینوکس یا گنو/لینوکس است.

---

[۷] GCC: GNU Compiler Collection

[۸] GNU Hurd

[۹] Micro Kernel

[۱۰] Linus Torvalds

[۱۱] Monolothic Kernel

[۱۲] Linux

فصل اول

کلیات

۱-۱) هدف

هدف این پروژه بررسی و تشریح روش ایجاد یک سیستم‌عامل کارا برمبنای فن‌آوری‌های آزاد است که در خلال آن، خواننده با مفاهیمی هم‌چون کامپایل کامپایلر برای مقاصد هدف گوناگون، قابل بوت کردن پارتیشن‌ها و... نیز آشنا می‌شود.

۱-۲) پیشینه‌ی کار و تحقیق

نخستین بار در سال ۱۹۸۴ میلادی بنیاد نرم‌افزار آزاد در پروژه‌ای به نام «گنو» شروع به طراحی و ساخت یک سیستم‌عامل آزاد نمود تا بتواند نرم‌افزار آزاد را بر روی آن اجرا کرده و یک اکوسیستم آزاد نرم‌افزاری را ایجاد کند که در آن برخلاف دیگر سامانه‌های مرسوم آن زمان، حقوق توسعه‌دهده و مصرف‌کننده به صورت توأمان حفظ شود و انحصار تکنولوژی‌های نوین از شرکت‌های بزرگی که به علّت داشتن منابع مالی، خود را صاحب امتیاز تمام محصولاتی می‌دانستند که تا آن زمان تنها در چنین شرکت‌هایی تولید می‌شد، برداشته شود.

در سال ۱۹۹۱ «لینوس تروالدز» یک هسته‌ی سیستم‌عامل از نوع یک‌پارچه را به صورت آزاد ارائه کرد که با همکاری اعضای پروژه‌ی گنو توانست درون این سیستم‌عامل جا گرفته و نقش هسته‌ی اصلیِ هنوز ناتمام، امّا کاراتر این پروژه را ایفا کند. به کمک این هسته که «لینوکس» نام‌گذاری شد، نخستین سیستم‌عامل آزاد تحت نام «گنو/لینوکس» شروع به کار نمود.

در واپسین سال هزاره‌ی دوم شخصی به نام «جرارد بیک‌منز»[1] تصمیم گرفت سیستم‌عامل آزاد خودش را آن‌گونه که می‌خواست از نو بسازد. وی که به نبود مستندات کافی برای این‌کار پی برده

[1] Gerard Beekmans

بود، مراحل انجام این کار و نتایجش را در کتابی به نام Linux From Scratch منتشر کرد که در میان هوادارانش به نام LFS شناخته می‌شود.

۱-۳) روش کار و تحقیق

برای انجام این پروژه، از سیستم‌عامل آرچ‌لینوکس با معماری i686 درون یک ماشین مجازی از نوع VirtualBox بر روی میزبان اوبونتو ۱۲.۰۴ با معماری x86_64 استفاده شد. روند کلّی ساخت سیستم‌عامل از کتاب LFS ایده گرفته شد. برای نگارش پایان‌نامه از ویرایش‌گر Gummi 0.6 و ویرایش ۲۰۰۹ از موتور تک XeLaTeXو ماکروهای XePersianبرروی اوبونتو ۱۲.۰۴ استفاده شد.

فصل دوم

آماده‌سازی برای ساخت

۲-۱) فراهم کردن یک پارتیشن جدید

در این بخش پارتیشنی که سیستم‌عامل جدید را میزبانی می‌کند ساخته می شود. ما ابتدا خود پارتیشن را خواهیم ساخت، سپس بر روی آن یک فایل‌سیستم ایجاد کرده و در انتها آن را سوار خواهیم کرد.

۲-۱-۱) ساخت یک پارتیشن جدید

مانند اکثر سیستم‌های عامل دیگر، سیستم‌عامل جدیدمعمولاً روی یک پارتیشن تخصیص یافته نصب می‌شود. روش پیشنهادی ساخت یک سیستم‌عامل جدیداستفاده از یک پارتیشن خالی موجود، یا در صورت داشتن فضای خالی بدون پارتیشین کافی، ساخت آن است.

یک سامانه‌ی کمینه نیاز به پارتیشنی حدود ۲/۸ گیگابایت(GB) دارد. این برای ذخیره‌ی همه‌ی تربال[۱]های منبع و کامپایل بسته‌ها کافی است. به هرحال اگر سیستم‌عامل جدیدقرار است به عنوان سامانه‌ی گنو/لینوکس ابتدایی استفاده شود، احتمالا باید نرم‌افزار اضافی‌ای نصب شود که نیاز به فضای اضافی دارد. یک پارتیشن ۱۰گیگابایتی فضای معقولی است. خود سیستم‌عامل جدیداین‌قدر جا نمی‌گیرد. مقدار زیادی از این نیازمندی برای فراهم کردن فضای ذخیره‌ی موقّتی کافی خالی است. کامپایل کردن بسته‌ها می‌تواند نیاز به مقدار زیادی فضای دیسک داشته باشد که بعد از نصب بسته خالی می‌شود.

چون همیشه حافظه‌ی اصلی کافی برای فرآیند کامپایل وجود ندارد، ایده‌ی خوبی است که از یک پارتیشن کوچک روی دیسک به عنوان فضای swap استفاده شود. این فضا توسّط هسته برای دخیره‌ی داده‌های کم‌تر مورد استفاده به کار گرفته می شود تا فضای بیش‌تری برای پروسه‌های فعّال وجود داشته باشد. پارتیشن swap برای سیستم‌عامل جدیدمی‌تواند با سامانه‌ی میزبان یکی باشد که

[۱] Tarball

در این صورت نیازی به ساخت پارتیشن جدیدی برای این کار نیست.

۲-۱-۲) ساخت فایل‌سیستم روی پارتیشن

پس از این که پارتیشن جدید ساخته شد، می‌توان روی آن فایل‌سیستم را ایجاد کرد. پراستفاده‌ترین فایل‌سیستم در دنیای گنو، «فایل‌سیستم توسعه‌پذیر چهارم» (Ext4)، فایل‌سیستمی با قابلیت‌های journaling است که می‌توان با دستور زیر آن را روی پارتیشن ایجاد کرد:

```
mke2fs -jv /dev/<xxx>
```

عبارت <xxx> باید با نام پارتیشنی که برای سیستم‌عامل جدیدساخته شد جایگزین شود.

اگر از پارتیشن swap موجود استفاده می‌شود نیازی به فرمت کردن آن نیست.اگر پارتیشن swap جدیدی ایجاد شده، باید با دستور زیر آن را راه‌اندازی اوّلیه کرد:

```
mkswap /dev/<yyy>
```

۲-۱-۳) سوار کردن پارتیشن جدید

حال که فایل سیستم ساخته شد باید بتوان به آن دست‌رسی پیدا کرد. برای این‌کار باید پارتیشن در یک نقطه‌ی اتّصال سوار شود. در این پایان‌نامه فرض می‌شود که فایل‌سیستم روی /mnt/lfs سوار شده است.

با این دستور نقطه‌ی اتّصال به متغیر محیطی LFS تخصیص داده می شود:

```
export LFS=/mnt/lfs
```

نقطه‌ی اتّصال با دستور زیر ساخته شده و فایل‌سیستم روی آن سوار می‌شود:

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
```

اگر از پارتیشن swap استفاده می‌شود با این دستور می‌توان آن را فعّال کرد:

```
swapon -v /dev/<yyy>
```

۲-۲) بسته‌ها و وصله‌ها

این بخش شامل سیاهه‌ی بسته‌هایی است که نیاز است برای ساخت یک سامانه‌ی گنو/لینوکس پایه بارگیری شوند. این پایان‌نامه با نسخه‌های گفته شده ی این نرم‌افزار ساخته شد و ممکن است با دیگر نسخه‌های آن‌ها کار نکند.

بسته‌ها و وصله‌های بارگیری شده باید در جایی ذخیره شوند که که در طول ساخت همواره دردست‌رس باشند. هم‌چنین یک شاخه‌ی فعّال برای بازکردن منبع‌ها و ساخت آن‌ها مورد نیاز است. /$LFS/sources می‌تواند هم به عنوان محل نگه‌داری تربال‌ها و وصله‌ها و هم به عنوان شاخه‌ی فعَال استفاده شود. با استفاده از این شاخه، المان‌های موردنیازروی پارتیشن LFS قرار گرفته و در تمام مراحل فرایند ساخت موجود هستند.

برای ساخت این شاخه، پیش از شروع نشست بارگیری باید این دستور به عنوان کاربر ریشه اجرا شود:

```
mkdir -v $LFS/sources
```

این شاخه باید قابل نوشتن و چسبناک شود. چسبناک بدین معناست که حتا اگر چندین کاربر اجازه‌ی نوشتن روی یک شاخه را دارند، تنها صاحب یک پرونده می‌تواند آن را حذف کند. دستور زیر حالت‌های نوشتن و چسبناک را فعّال می‌کند:

```
chmod -v a+wt $LFS/sources
```

۲-۲-۱) بسته‌های مورد استفاده

- Autoconf (2.68) - 1,350 KB

این بسته شامل برنامه‌هایی برای تولید اسکریپت‌های پوسته‌ای است که می‌توانند به طور خودکار کد منبع را از یک الگوی توسعه‌دهنده پیکربندی کنند. این معمولاً برای بازسازی یک بسته بعد از به‌روزرسانی روال‌های ساخت نیاز است.

- Automake (1.11.3) - 1,051 KB

این بسته شامل برنامه‌هایی برای تولید پرونده‌های ساخت از یک الگو است. این معمولاً برای بازسازی یک بسته بعد از به‌روزرسانی روال‌های ساخت نیاز است.

- Bash (4.2) - 6,845 KB

این بسته نیازمندی هسته‌ی LSB را برای فراهم اوردن یک محیط پوسته‌ی Bourne برای سامانه تأمین می‌کند. این به خاطر استفاده‌ی مرسومش و توانایی گسترشش در توابع پوسته‌ای ساده در میان دیگر بسته‌های پوسته برگزیده شد.

- Binutils (2.22) - 19,505 KB

این بسته شامل یک پیونددهنده، یک اسمبلر، و دیگر ابزارها برای handleکردن پرونده‌های شئ است. برنامه‌های درون این بسته برای کامپایل بیش‌تر بسته‌ها نیاز هستند.

- Bison (2.5) - 1,983 KB

این بسته شامل نسخه‌ی گنوی yacc[۲] است که برای ساخت خیلی از دیگر برنامه‌های سیستم‌عامل جدیدمورد نیاز است.

- Bzip2 (1.0.6) - 764 KB

این بسته شامل برنامه‌هایی برای فشرده‌سازی و باز کردن پرونده‌هاست. این برای باز کردن خیلی از بسته‌های سیستم‌عامل جدیدمورد نیاز است.

- Check (0.9.8) - 546 KB

این بسته شامل رابطی برای چارچوب بررسی واحد به زبان C است.

- Coreutils (8.15) - 4,827 KB

این بسته شامل تعدادی برنامه‌ی ضروری برای مشاهده و دست‌کاری پرونده‌ها و شاخه‌هاست.

---

[۲] Yet Another Compiler Compiler

- DejaGNU (1.5) - 563 KB

این بسته شامل چارچوبی برای آزمودن دیگر برنامه‌هاست. این فقط روی زنجیرابزار موقّت نصب می‌شود.

- Diffutils (3.2) - 1,976 KB

این بسته شامل برنامه‌هایی است که تفاوت‌های بین پرونده‌ها و شاخه‌ها را نشان می‌دهند. این برنامه‌ها می‌توانند برای ساخت وصله‌ها استغاده شوند، و هم‌چنین در روال‌های ساخت خیلی از بسته‌ها استفاده می‌شوند.

- E2fsprogs (1.42) - 5,576 KB

این بسته شامل ابزارهایی برای ساخت فایل‌سیستم‌های توسعه‌پذیر است.

- Expect (5.45) - 614 KB

این بسته شامل برنامه‌ای برای carry outکردن گفت‌وگوهای اسکریپتی با دیگر برنامه‌های interactive است. این به طور معمول برای آزمودن دیگر بسته‌ها استفاده می‌شود. این فقط روی زنجیرابزار موقّت نصب می‌شود.

- File (5.10) - 595 KB

این بسته شامل ابزاری برای تشخیص نوع پرونده یا پرونده‌های داده‌شده است. چند بسته برای ساخت به این نیاز دارند.

- Findutils (4.4.2) - 2,100 KB

این بسته شامل برنامه‌هایی برای یافتن پرونده‌ها در یک سیتم پرونده است. این در اسکرپت ساخت چندین بسته استفاده می‌شود.

- Flex (2.5.35) - 1,227 KB

این بسته شامل ابزاری برای تولید برنامه‌هایی است که الگوها را درمتن تشخیص می‌دهند. این

نسخه‌ی گنوی برنامه‌ی lex[۳] است. این برای ساخت خیلی از بسته‌های سیستم‌عامل جدیدنیاز است.

- Gawk (4.0.0) - 2,016 KB

این بسته شامل برنامه‌هایی برای دست‌کاری پرونده‌های متنی است. این نسخه‌ی گنوی awk[۴] است. این در اسکریپت ساخت خیلی از بسته‌های دیگر استفاده می‌شود.

- GCC (4.6.2) - 70,308 KB

این بسته مجموعه کامپایلر گنو[۵] است. این شامل کامپایلرهای C، C++ و خیلی از زبان‌های دیگری که توسط سیستم‌عامل جدیدساخته نمی‌شوند می‌شود.

- GDBM (1.10) - 640 KB

این بسته شامل کتاب‌خانه‌ی مدیریت پایگاه داده‌ی گنو[۶] می‌باشد. این توسط یکی دیگر از بسته‌های سیستم‌عامل جدید(Man-DB) استفاده می‌شود.

- Gettext (0.18.1.1) - 14,785 KB

این بسته شامل ابزارها و کتاب‌خانه‌هایی برای بین‌المللی‌سازی و محلّی‌سازی شماری از بسته‌هاست.

- Glibc (2.14.1) - 15,284 KB

این بسته شامل کتاب‌خانه‌ی اصلی C است. برنامه‌های گنو/لینوکس بدون این اجرا نخواهند شد.

---

[۳] Lexical analyzer

[۴] Aho-Weinberg-Kernighan

[۵] GNU Compiler Collection

[۶] GNU DataBase Manager library

- GMP (5.0.4) - 1,650 KB

این بسته شامل کتاب‌خانه‌های ریاضی است و توابع مفیدی را برای محاسبات با دقّت دل‌خواه فراهم می‌کند. این برای ساخت GCC نیاز است.

- Grep (2.10) - 1,048 KB

این بسته شامل برنامه‌هایی برای جست‌وجو در میان پرونده‌هاست. این توسط بیش‌تر اسکریپت‌های ساخت استفاده می‌شود.

- Groff (1.21) - 3,774 KB

این بسته شامل برنامه‌هایی برای پردازش و شکل‌دهی به متن است. یک تابع مهم این برنامه‌ها برای شکل‌دهی صفحات راهنماست.

- GRUB (1.99) - 4,544 KB

این بسته بارگذار راه‌انداز عظیم متّحد [۷] است. این یکی از چندین بارگذار راه‌انداز موجود، ولی منعطف‌ترین آن‌هاست.

- Gzip (1.4) - 886 KB

این بسته شامل برنامه‌هایی برای فشرده‌سازی و باز کردن پرونده‌هاست. این برای باز کردن خیلی از بسته‌های سیستم‌عامل جدیدمورد نیاز است.

- Iana-Etc (2.30) - 201 KB

این بسته برای خدمات و پروتکل‌های شبکه داده فراهم می‌کند. این برای فعال کردن قابلیت‌های شبکه‌ای درست لازم است.

- Inetutils (1.9.1) - 1,941 KB

این بسته شامل برنامه‌هایی برای مدیریت پایه‌ای شبکه است.

---

[۷] Grand Unified Bootloader

- IPRoute2 (3.2.0) - 365 KB

این بسته شامل برنامه‌هایی برای شبکه‌سازی پایه‌ای و پیش‌رفته‌ی IPv4 و IPv6 است. این به خاطر قابلیت‌های بیش‌تر IPv6اش از دیگر بسته‌ی ابزارهای شبکه‌ی مرسوم (net-tools) انتخاب شد.

- Kbd (1.15.2) - 1,520 KB

این بسته شامل پرونده‌ی جدول کلیدها، ابزارهای صفحه‌کلید برای صفحه‌کلیدهای غیر امریکایی، و تعدادی فونت کنسول است.

- Kmod (5) - 855 KB

این بسته شامل یک ماژول کرنل است که حاوی کدهایی است در هنگام اجرای کرنل توسعه یافته و رشد می‌کنند.

- Less (444) - 301 KB

این بسته شامل مشاهده‌گر متن خیلی خوبی است که اجازه‌ی لغزش به بالا و پایین را هنگام مشاهده‌ی یک پرونده می‌دهد. این نیز هم‌چنین توسط Man-DB برای مشاهده ی صفحات راهنما به کار می‌رود.

- LFS-Bootscripts (20120229) - 32 KB

این بسته شامل اسکریپت‌هایی برای بهینه ساختن فرآیند راه‌اندازی سامانه است.

- Libpipeline (1.2.0) - 670 KB

این بسته شامل کتاب‌خانه‌ای برای تنظیم و اجرای خط‌لوله‌ی پردازش‌هاست.

- Libtool (2.4.2) - 2,571 KB

این بسته شامل اسکریپت پشتیبانی کتاب‌خانه‌ی ژنریک گنو است. این پیچیدگی استفاده از کتاب‌خانه‌های مشترک را در یک سازگاری پنهان می‌کند.

- Linux (3.2.6) - 63,560 KB

این بسته رابط سخت‌افزار و نرم‌افزار است. این، «لینوکس» در سیستم‌عامل گنو/لینوکس است.

- M4 (1.4.16) - 1,229 KB

این بسته شامل یک پردازنده‌ی macro متن عمومی است که به‌عنوان یک ابزار ساخت برای دیگر برنامه‌ها مفید است.

- Make (3.82) - 1,21 KB

این بسته شامل برنامه‌ای برای یک‌سره کردن ساخت بسته‌هاست. این توسط تقریباً هر بسته‌ای در سیستم‌عامل جدیدمورد نیاز است.

- Man-DB (2.6.1) - 2,449 KB

این بسته شامل برنامه‌هایی برای یافتن و مشاهده‌ی صفحات راهنما است. این به خاطر قابلیت‌های بین‌المللی‌سازی فوق‌العاده‌اش به‌جای man انتخاب شد. البته این برنامه، وظایف man را نیز انجام می‌دهد.

- Man-pages (3.35) - 1,650 KB

این بسته شامل محتوای واقعی صفحات راهنمای پایه‌ی لینوکس است.

- MPC (0.9) - 553 KB

این بسته شامل توابعی برای حساب اعداد مختلط است. این برای GCC مورد نیاز است.

- MPFR (3.1.0) - 1,176 KB

این بسته شامل توابعی برای حساب با دقت چندگانه است. این برای GCC مورد نیاز است.

- Ncurses (5.9) - 2,760 KB

این بسته شامل کتاب‌خانه‌هایی برای handle کردن مستقل از پایانه‌ی صفحات نویسه است.

این معمولا برای فراهم آوردن کنترل مکان‌نما برای یک پایانه‌ی فهرست‌بندی استفاده می‌شود. این برای تعدادی از بسته‌ها در سیستم‌عامل جدیدنیاز است.

- Patch (2.6.1) - 248 KB

این بسته شامل برنامه‌ای برای تغییر یا ایجاد پرونده‌ها توسط اعمال یک پرونده‌ی وصله است که به طور معمول توسط برنامه‌ی diff ایجاد می‌شود. این برای روال ساخت چندین بسته در سیستم‌عامل جدیدمورد نیاز است.

- Perl (5.14.2) - 12,917 KB

این بسته یک مفسر برای زبان زمان اجرای PERL است. این برای نصب و آزمایش خیلی از بسته‌های سیستم‌عامل جدیدنیاز است.

- Procps (3.2.8) - 279 KB

این بسته شامل برنامه‌هایی برای نظارت بر پردازه‌هاست. این برنامه‌ها برای مدیریت سامانه مفیدند، و هم‌چنین توسط اسکریپت‌های Boot سیستم‌عامل جدیداستفاده می‌شوند.

- Psmisc (22.15) - 382 KB

این بسته شامل برنامه‌هایی برای نمایش اطلاعات درباره‌ی پردازه‌های درحال اجرا است. این برنامه‌ها برای مدیریت سامانه مفیدند.

- Readline (6.2) - 2,225 KB

این بسته مجموعه ای از کتاب‌خانه‌هاست که ویرایش خط فرمان و قابلیت‌های تاریخچه را offer. این توسط Bash استفاده می‌شود.

- Sed (4.2.1) - 878 KB

این بسته ویرایش متن را بدون باز کردن آن در یک ویرایش‌گر متن مجاز می‌کند. این هم‌چنین برای اسکریپت‌های پیکربندی بیش‌تر بسته‌های سیستم‌عامل جدیدنیاز است.

- Shadow (4.1.5) - 2,105 KB

این بسته شامل برنامه‌هایی برای handle گذرواژه‌ها به شیوه‌ای امن است.

- Sysklogd (1.5) - 85 KB

این بسته شامل برنامه‌هایی برای گزارش‌گیری از پیام‌های سامانه، مانند آن‌هایی است که توسط کرنل داده می‌شودیا پردازه‌های daemon وقتی اتفاق غیرمعمولی رخ می‌دهد.

- Sysvinit (2.88dsf) - 108 KB

این بسته برنامه‌ی init را که والد تمامی پردازه‌های دیگر روی سامانه‌ی لینوکسی است، فراهم می‌کند.

- Tar (1.26) - 2,285 KB

این بسته قابلیت‌های بایگانی و extraction تقریباً تمامی بسته‌های مورد استفاده در سیستم‌عامل جدیدرا فراهم می‌کند.

- Tcl (8.5.11) - 4,379 KB

این بسته شامل زبان دستور ابزار[۸] است که در بسیاری از مجوعه آزمایش‌های بسته‌های سیستم‌عامل جدیداستفاده می‌شود. این فقط روی زنجیرابزار موقّت نصب می‌شود.

- Texinfo (4.13a) - 2,687 KB

این بسته شامل برنامه‌هایی برای خواندن، نوشتن، و تبدیل صفحات اطلاعات است. این در روال‌های نصب بسیاری از بسته‌های سیستم‌عامل جدیداستفاده می‌شود.

- Udev (181) - 678 KB

این بسته شامل برنامه‌هایی برای ساخت پویای گره‌های قطعه است. این جایگزینی برای ایجاد هزاران قطعه‌ی ایستا در شاخه‌ی dev/ است.

---

[۸] Tool Command Language

- Udev-config (20100128) - 7 KB

این بسته شامل اسکریپتی برای پیکر بندی Udev است.

- Util-linux (2.20.1) - 4,506 KB

این بسته شامل برنامه‌های سیستمی متفرقه است. در میان آن‌ها ابزارهایی برای handling سیستم‌های پرونده، کنسول‌ها، پارتیشن‌ها و پیام‌ها وجود دارد.

- Vim (7.3) - 8,675 KB

این بسته شامل یک ویرایش‌گر است. این به‌خاطر سازگاریش با ویرایش‌گر vi کلاسیک و شمار عظیم قابلیت‌های قدرت‌مندش انتخاب شد. ویرایش‌گر برای بسیاری از افراد یک انتخاب خیلی شخصی است و هر ویرایش‌گر دیگری می‌تواند به دلخواه جایگزین شود.

- Xz Utils (5.0.3) - 1,002 KB

این بسته شامل برنامه‌هایی برای فشرده‌سازی و باز کردن پرونده‌هاست. این بیش‌ترین فشرده‌سازی عمومی در دست‌رس را فراهم می‌کند و برای باز کردن بسته‌هایی با فرمت XZ یا LZMA مفید است.

- Zlib (1.2.6) - 490 KB

این بسته شامل روال‌های فشرده‌سازی و باز کردن است که توسط برخی برنامه‌هااستفاده می‌شود.

۲-۲-۲) وصله‌های مورد نیاز

علاوه بر بسته‌ها، هم‌چنین چند وصله نیز مورد نیاز هستند. این وصله‌ها اشتباهاتی در بسته‌ها که باید توسط نگه‌دارنده درست می‌شد را تصحیح می‌کنند. این بسته‌ها هم‌چنین تغییرات کوچکی را برای راحت‌تر کردن کار با بسته‌ها ایجاد می‌کنند. لیست وصله‌های مورد نیاز در پیوست ۱ آمده است.

۲-۳) آماده‌سازی‌های نهایی

۲-۳-۱) درباره‌ی $LFS

در طول این پایان‌نامه متغیّر محیطی LFS استفاده می‌شود. ضروری است که این متغیّر همواره تعریف‌شده و روی نقطه‌ی اتّصال انتخاب شده برای پارتیشن LFS تنظیم شده باشد.

۲-۳-۲) ایجاد شاخه‌ی $LFS/tools

همه‌ی برنامه‌هایی که در بخش ۲-۴ کامپایل می‌شوند، در $LFS/tools نصب می‌شوندتا از برنامه‌هایی که در فصل ۳ کامپایل می‌شوند جدا شوند. برنامه‌هایی که این‌جا کامپایل می‌شوند ابزارهای موقّت هستند و بخشی از سیستم‌عامل جدیدنخواهند بود. با نگه داشتن این برنامه‌ها در یک شاخه‌ی جدا، آن‌ها می‌توانند پس از استفاده به راحتی دور انداخته شوند.

با اجرای دستور زیر به عنوان کاربر ریشه شاخه‌ی مورد نیاز ساخته خواهد شد:

```
mkdir -v $LFS/tools
```

گام بعدی ساخت یک پیوند سیستمی tools/ روس سیستم میزبان است که به شاخه‌ی تازه ساخته شده روی پارتیشن LFS اشاره کند تا زنجیرابزار بتواند کامپایل شود:

```
ln -sv $LFS/tools /
```

۲-۳-۳) افزودن کاربر LFS

هنگام ورود به عنوان کاربر ریشه، کوچک‌ترین اشتباهی می‌تواند به سامانه ضربه زده یا آن را نابود کند. بنابراین بهتر است بسته‌های این بخش توسط یک کاربر بدون دست‌رسی ساخته شوند. اجرای دستورات زیر به عنوان کاربر ریشه، کاربری فرضی به نام lfs عضوی از گروه lfs ایجاد می‌کند:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

برای ورود به سیستم به عنوان lfs باید گذرواژه‌ای را به آن اختصاص داد:

```
passwd lfs
```

با تغییر مالکیت شاخه‌ها به lfs، این کاربر به آن‌ها دست‌رسی کامل خواهد داشت:

```
chown -v lfs $LFS/sources
chown -v lfs $LFS/tools
```

برای ورود به سیستم به عنوان کاربر lfs می‌توان به شیوه ی زیر عمل کرد:

```
su - lfs
```

۲-۳-۴) تنظیم کردن محیط

برای داشتن یک محیط کاری خوب می‌توان پرونده‌های bash_profile. و bashrc. را ایجاد و تنظیم کرد. برای ایجاد پرونده‌ی bash_profile. می‌توان بدین صورت عمل کرد:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
EOF
```

و پرونده‌ی bashrc. نیز می‌تواند مقادیر زیر را داشته باشد:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL LFS_TGT PATH
EOF
```

پس از تنظیم پرونده‌های بالا، باید دستور زیر را برای اعمال تغییرات گفته شده در نشست جاری وارد کرد تا محیط برای ساخت ابزارهای موقّت کاملاً آماده شود:

```
source ~/.bash_profile
```

۲-۳-۵) درباره‌ی SBUها

از آن‌جایی که بسته‌ها روی سامانه‌های متفاوتی کامپایل می‌شوند، تعیین مدّت زمان مشخصی برای طول کامپایل شدن آن‌ها غیرممکن است. بزرگ‌ترین این بسته‌ها (Glibc) روی سریع‌ترین سامانه‌ها تقریباً به ۲۰ دقیقه زمان نیاز دارد، امّا ممکن است روی سامانه‌های ضعیف‌تر تا سه روز هم طول

بکشد! از این رو به‌جای استفاده از واحدهای زمانی واقعی، از واحد استاندارد ساخت[۹] یا SBU استفاده می‌کنیم.

روش محاسبه‌ی SBU بدین صورت است که مدّت زمان کامپایل شدن نخستین بسته (در این‌جا Binutils) را به عنوان یک واحد درنظر گرفته و بقیه‌ی بسته‌ها نسبت به آن سنجیده می‌شوند. در حالت کلّی SBU چندان دقیق نیست، زیرا به عوامل مختلفی مثل نسخه‌ی GCC سامانه‌ی میزبان و... بستگی دارد، امّا می‌تواند درک تقریبی خوبی از مدّت زمان کامپایل بدهد.

۲-۳-۶) درباره‌ی مجموعه‌های آزمایشی

بیش‌تر بسته‌ها یک مجموعه‌ی ازمایشی به همراه دارند که می‌تواند بررسی کند آیا همه‌چیز به درستی کامپایل شده است یا خیر. اگر مجموعه‌ی آزمایشی‌ای بتواند در بررسی‌ها قبول شود معمولاً بدین معناست که آن‌گونه که توسعه‌دهنده انتظار داشته عمل می‌کند، با این‌حال تضمینی بر این نیست که که بسته به طور کامل بدون اشکال است.

برخی مجموعه‌های آزمایشی از بقیه مهم‌تر هستند. برای مثال، مجموعه‌ی آزمایشی بسته‌های هسته‌ای زنجیرابزار –GCC، Binutils و Glibc– به خاطر نقش مرکزی‌شان در سامانه‌ای که به درستی کار کند، از بیش‌ترین اهمیّت برخوردارند. تکمیل مجموعه‌های آزمایشی GCC و Glibc می‌تواند مدّت زیادی طول بکشند، مخصوصاً روی سامانه‌های کندتر، امّا به شدّت توصیه می‌شود.

۲-۴) بنا کردن یک سامانه‌ی موقّتی

این بخش شیوه‌ی ساخت یک سامانه‌ی گنو/لینوکسی کمینه را نشان می‌دهد. این سامانه تنها ابزارهای کافی برای شروع ساخت سیستم‌عامل نهایی در فصل ۳ را دربر خواهد داشت.

در ساخت این سامانه‌ی کمینه دو مرحله وجود دارد. مرحله‌ی نخست ساخت یک زنجیرابزار

[۹] Standard Build Unit

(کامپایلر، اسمبلر، لینکر، کتاب‌خانه‌ها و تعدادی ابزار سیستمی مفید) جدید و مستقل از میزبان است. مرحله‌ی دوم از این زنجیرابزار برای ساخت دیگر ابزارهای ضروری استفاده می‌کند.

پرونده‌های کامپایل شده در این فصل در شاخه‌ی $LFS/tools نصب می‌شوند تا از پرونده‌های نصب شده در فصل بعدی جدا جدا بمانند، زیرا که بسته‌های کامپایل‌شده در این‌جا موقّتی هستند.

۲-۴-۱) نکات فنّی زنجیرابزار

پیش‌تر با تعیین مقدار متغّیر LFS_TGT این اطمینان حاصل شد که نخستین ساخت Binutils و GCC، کراس‌لینکر و کراس‌کامپایلری سازگار تولید می‌کند که به‌جای تولید کد دودویی برای معماری دیگر، کد دودویی سازگار با سخت‌افزار جاری تولید می‌کند.

کتاب‌خانه‌های موقّتی کراس‌کامپایل شده هستند، زیرا کراس‌کامپایلر بنا به طبیعتش نمی‌تواند به چیزی جز سامانه‌ی میزبان تکیه کند. کراس‌کامپایلر هم‌چنین امکان ساخت کتاب‌خانه‌های ۳۲بیتی و ۶۴بیتی را روی سخت‌افزار با توانایی ۶۴بیتی ممکن می‌سازد. تغییر با احتیاط پرونده‌های spec برای GCC به کامپایلر می‌گوید کدام لینکر پویای مقصد استفاده خواهد شد.

ابتدا Binutils نصب می‌شود زیرا که configure، هم GCC و هم Glibc را وادار می‌کند آزمون ویژگی‌های مختلفی را روی اسمبلر و لینکر انجام دهند تا دریابند کدام ویژگی‌های نرم‌افزار را فعّال یا غیرفعّال کنند. این از چیزی که ممکن است در ابتدا به نظر بیاید مهم‌تر است. یک GCC یا Glibc درست پیکربندی نشده می‌تواند منجر به یک زنجیرابزار خراب شود که نتایج اشتباهاتش تا نزدیکی‌های پایان ساخت یک توزیع کامل نیز محرز نشود. یک اشتباه در مجموعه آزمایشی معمولاً این خطا را پیش از انجام کارهای اضافی مشخّص می‌کند.

بسته‌ی بعدی که نصب می‌شود GCC است. مثالی از آن‌چه می‌تواند به هنگام اجرای configure آن دیده شود این است:

```
checking what assembler to use... /tools/i686-lfs-linux-gnu/bin/as
checking what linker to use... /tools/i686-lfs-linux-gnu/bin/ld
```

اطّلاعات دقیق‌تر می‌تواند توسط دادن گزینه‌ی v- هنگام کامپایل یک برنامه به GCC به‌دست آید.

بسته‌ی بعدی که نصب می‌شود Glibc است. مهم‌ترین پیش‌نیازهای ساخت Glibc کامپایلر، ابزارهای دودویی، و سرآیندهای هسته هستند. از آن‌جا که Glibc همیشه از کامپایلر مربوط به پارامتر host- - که به اسکریپت پیکربندی‌اش فرستاده می‌شود(در این‌جا i686-lfs-linux-gnu-gcc) استفاده می‌کند، معمولاً مشکلی در مورد کامپایلر وجود ندارد. ابزارهای دودویی و سرآیندهای هسته ممکن است کمی پیچیده‌تر باشند. نبابراین ریسک نکرده و برای اطمینان از انتخاب‌های صحیح از سوئیچ‌های پیکربندی موجود استفاده می‌شود.

پس از نصب Glibc، پرونده‌ی مشخّصات GCC را برای اشاره به لینکر پویای جدید در /tools/lib تغییر می‌دهیم. این آخرین حرکت برای اطمینان از این که جست‌وجو و پیوند تنها در پیش‌وند /tools انجام می‌شوند ضروری است.

برای گذر دوم GCC، نیاز است منبعش برای گفتن این که از لینکر پویای جدید استفاده کند، تغییر کند. انجام ندادن این عمل موجب می‌شود برنامه‌های GCC نام لینکر پویای شاخه‌ی /lib سیستم میزبان را درون خود داشته باشند، که ما را از هدف خلاص شدن از میزبان دور می‌کند.

در طول گذر دوم Binutils، قادریم برای کنترل مسیر جست‌وجوی کتاب‌خانه‌ی ld، سوئیچ پیکربندی with-lib-path- - را به‌کار ببریم. از این نقطه به بعد، زنجیرابزار مرکزی خودشمول و خودمیزبان می‌شود. باقی‌مانده‌ی بسته‌های این فصل توسّط Glibc جدید در /tools ساخته می‌شوند.

به‌محض ورود به محیط chroot در فصل ۳، نخستین بسته‌ی اصلی که نصب می‌شود Glibc است، به خاطر خودبسندگی ذاتی‌اش که در بالا ذکر شد. پس از این که این Glibc در /usr نصب شد، یک جابه‌جایی سریع در پیش‌فرض‌های زنجیرابزار انجام می‌دهیم، و بعد اقدام به ساخت ادامه‌ی سیستم‌عامل جدیدمی‌کنیم.

۲-۴-۲) دستورالعمل عمومی کامپایل کردن

۱. همه‌ی منابع و وصله‌ها در یک شاخه که از محیط chroot قابل دست‌رسی باشد قرار می‌گیرند، مانند /$LFS/sources. منابع نباید در /$LFS/tools قرار بگیرند.

۲. به شاخه‌ی منبع می‌رویم

۳. برای هر بسته:

(الف) با استفاده از برنامه‌ی tar، بسته از حالت فشرده خارج می‌شود.

(ب) به شاخه‌ی ساخته شده هنگام خروج بسته از حالت فشرده می‌رویم.

(ج) دستورالعمل ساخت بسته را دنبال می‌کنیم.

(د) به شاخه‌ی منابع بازمی‌گردیم.

(ه) شاخه‌ی منبع از حالت فشرده خارج شده و هر شاخه‌ی <package>-build‌ای که در فرایند ساخت ایجاد شد را پاک می‌کنیم، مگر این که دستورالعمل چیز دیگری گفته باشد.

۲-۴-۳) Binutils – گذر نخست

زمان تقریبی ساخت: 1 SBU

فضای دیسک مورد نیاز: 350 MB

مستندات این بسته توصیه می‌کند Binutils بیرون از شاخه‌ی منبع و در یک شاخه‌ی ساخت تخصیص داده شده ساخته شود:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

با این دستور Binutils برای کامپایل آماده می‌شود:

```
../binutils-2.22/configure \
--target=$LFS_TGT --prefix=/tools \
--disable-nls --disable-werror
```

و این گونه بسته کامپایل می‌شود:

```
make
```

در حال عادی اکنون مجموعه‌ی آزمایشی اجرا می‌شود، ولی در این مرحله‌ی ابتدایی چارچوب مجموعه‌ی آزمایشی(TCL، Expect و DejaGNU) هنوز سرجایش نیست. سود اجرای آزمایش‌ها در این نقطه کمینه است، زیرا که برنامه‌های این گذر نخست به‌زودی با برنامه‌های گذر دوم عوض خواهند شد.

با دستور زیر این بسته نصب می‌شود:

```
make install
```

۲-۴-۴) GCC - گذر نخست

زمان تقریبی ساخت: 5 SBU

فضای دیسک مورد نیاز: 1.2 GB

GCC اکنون نیازمند بسته‌های GMP، MPFR و MPC است. از آن‌جا که ممکن است این بسته‌ها در توزیع میزبان وجود نداشته باشند، توسط GCC ساخته می‌شوند. هربسته درون شاخه‌ی منبع GCC بازگشایی می‌شود و شاخه‌های حاصل تغییرنام پیدا می‌کنند تا روال‌های ساخت GCC به صورت خودکار از آن‌ها استفاده کنند:

```
tar -jxf ../mpfr-3.1.0.tar.bz2
mv -v mpfr-3.1.0 mpfr
tar -Jxf ../gmp-5.0.4.tar.xz
mv -v gmp-5.0.4 gmp
tar -zxf ../mpc-0.9.tar.gz
mv -v mpc-0.9 mpc
```

در این‌جا وصله‌ای اعمال می‌شود که اجازه می‌دهد ساخت کتاب‌خانه‌های هدف libiberty و zlib غیرفعّال شود، زیرا که در یک محیط کراس‌کامپایل شده درست ساخته نمی‌شوند:

```
patch -Np1 -i ../gcc-4.6.2-cross_compile-1.patch
```

مستندات این بسته توصیه می‌کند GCC بیرون از شاخه‌ی منبع و در یک شاخه‌ی ساخت تخصیص داده شده ساخته شود:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

با این دستور GCC برای کامپایل آماده می‌شود:

```
../gcc-4.6.2/configure \
--target=$LFS_TGT --prefix=/tools \
--disable-nls --disable-shared --disable-multilib \
--disable-decimal-float --disable-threads \
--disable-libmudflap --disable-libssp \
--disable-libgomp --disable-libquadmath \
--disable-target-libiberty --disable-target-zlib \
--enable-languages=c --without-ppl --without-cloog \
--with-mpfr-include=$(pwd)/../gcc-4.6.2/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs
```

و این گونه بسته کامپایل می‌شود:

```
make
```

در حال عادی اکنون مجموعه‌ی آزمایشی اجرا می‌شود، ولی همان‌گونه که پیش‌تر اشاره شد، چارچوب مجموعه‌ی آزمایشی هنوز سرجایش نیست. سود اجرای آزمایش‌ها در این نقطه کمینه است، زیرا که برنامه‌های این گذر نخست به‌زودی عوض خواهند شد.

با دستور زیر این بسته نصب می‌شود:

```
make install
```

استفاده از diable-shared- - بدین معناست که پرونده‌ی libgcc_eh.a ساخته و نصب نشده است. بسته‌ی Glibc به‌خاطر استفاده از lgcc_eh- در سامانه‌ی ساختش، به این کتاب‌خانه وابسته است. این نیازمندی می‌تواند توسّط ساخت یک پیوند نرم به libgcc.a ارضا شود، زیرا آن پرونده قرار است شامل چیزهایی شود که معمولا در libgcc_eh.a وجود دارد:

```
ln -vs libgcc.a `$LFS_TGT-gcc -print-libgcc-file-name | \
sed 's/libgcc/&_eh/'`
```

۲-۴-۵) سرآیندهای API لینوکس

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 511 MB

هسته‌ی لینوکس باید برای استفاده‌ی کتاب‌خانه‌ی C سامانه (در این‌جا Glibc) یک رابط برنامه‌نویسی نرم‌افزار فراهم کند.این کار با نصب سرآیندهای C مختلفی که همراه با تربال منبع هسته‌ی لینوکس ارائه می‌شوند امکان‌پذیر است.

ابتدا باید مطمئن شد پرونده و پیش‌نیازی از فعّالیت‌های پیشین باقی نمانده است:

```
make mrproper
```

حال می‌توان سرآیندهای قابل مشاهده برای کاربر هسته را از منبع آزمایش کرد و از حالت فشرده درآورد. آن‌ها در یک شاخه‌ی محلّی بی‌درنگ قرار دارند و در مکان مورد نیاز رونوشت شده‌اند، زیرا که فرایند باز کردن از حالت فشرده هر پرونده‌ی موجودی در شاخه‌ی مقصد را برمی‌دارد.

```
make headers_check
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /tools/include
```

Glibc (۶-۴-۲

زمان تقریبی ساخت: 5.5 SBU

فضای دیسک مورد نیاز: 501 MB

بسته‌ی Glibc شامل کتاب‌خانه‌ی اصلی C است. این کتاب‌خانه روال‌های پایه‌ای را برای تخصیص حافظه، جست‌وجوی شاخه‌ها، باز کردن و بستن پرونده‌ها، خواندن و نوشتن پرونده‌ها، اداره‌ی رشته‌ها، تطبیق الگوها، محاسبه و مانند این‌ها فراهم می‌کند.

برای تصحیح باگی که از ساخته‌شدن Glibc توسّط GCC-4.6.2 جلوگیری می‌کند:

```
patch -Np1 -i ../glibc-2.14.1-gcc_fix-1.patch
```

هم‌چنین باید به یک بررسی سرآیند که به خاطر محیط ساخت ناکامل در این نقطه خطا می‌دهد نشانی داد:

```
patch -Np1 -i ../glibc-2.14.1-cpuid-1.patch
```

مستندات Glibc ساخت Glibc را در یک شاخه‌ی ساخت تخصیص داده شده در بیرون شاخه‌ی منبع توصیه می‌کنند:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

از آن‌جا که Glibc دیگر i386 را پشتیبانی نمی‌کند، توسعه‌دهندگانش می‌گویند که هنگام ساختنش برای ماشین‌های x86 از پرچم کامپایلر -march=i486 استفاده شود. برای انجام این کار راه‌های بسیاری وجود دارد، ولی بررسی‌ها نشان می‌دهند که بهترین‌جا برای قرار دادن این پرچم درون متغیّر ساخت «CFLAGS» است. به‌جای باطل کردن کامل آن‌چه که سامانه‌ی ساخت داخلی Glibc برای CFLAGS استفاده می‌کند، پرچم جدید را با استفاده از پرونده‌ی ویژه‌ی configparmsبه محتویات موجود CFLAGS می‌افزاییم. هم چنین پرچم -mtune-native برای بازتنظیم یک مقدار معقول برای -mtune که هنگام تنظیم کردن -march تغییر می‌کند لازم است.

```
case `uname -m` in
i?86) echo "CFLAGS += -march=i486 -mtune=native" > configparms ;;
esac
```

سپس Glibc برای کامپایل آماده می‌شود:

```
../glibc-2.14.1/configure --prefix=/tools \
--host=$LFS_TGT --build=$(../glibc-2.14.1/scripts/config.guess) \
--disable-profile --enable-add-ons \
--enable-kernel=2.6.25 --with-headers=/tools/include \
libc_cv_forced_unwind=yes libc_cv_c_cleanup=yes
```

و با این دستوز عملیات کامپایل انجام می‌پذیرد:

```
make
```

این بسته با یک مجموعه آزمایشی می‌آید که در حال حاضر قابل اجرا نیست، زیرا که ما هنوز یک کامپایلر C++ نداریم.

با این دستور بسته نصب می‌شود:

```
make install
```

۲-۴-۷) تنظیم کردن زنجیرابزار

حال که کتاب‌خانه‌های موقّتی C نصب شده‌اند، همه‌ی ابزارهایی که در ادامه‌ی این بخش کامپایل می‌شوند باید به این کتاب‌خانه‌ها لینک شوند. برای انجام این کار، باید پرونده‌ی مشخّصات

کراس‌کامپایلر طوری تنظیم شود که به لینکر پویای جدید در tools/ اشاره کند.

این‌کار با نسخه‌برداری از پرونده‌ی «specs» کامپایلر به مکانی که به صورت پیش‌فرض به دنبال آن بگردد انجام می‌پذیرد. یک جانشینی ساده‌ی sed لینکری که GCC استفاده می‌کند را تعویض می‌کند. قاعده در این‌جا یافتن همه‌ی ارجاع‌ها به پرونده‌ی لینکر پویا در lib/ یا در صورتی که سامانه‌ی میزبان امکان ۶۴بیت دارد lib64/ و تنظیم آن‌ها به گونه‌ای است که به مکان جدید در tools/ اشاره کند:

```
SPECS=`dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/specs
$LFS_TGT-gcc -dumpspecs | sed \
-e 's@/lib\(64\)\?/ld@/tools&@g' \
-e "/^\*cpp:$/{n;s,$, -isystem /tools/include,}" > $SPECS
echo "New specs file is: $SPECS"
unset SPECS
```

۲-۴-۸) Binutils – گذر دوم

زمان تقریبی ساخت: 1.1 SBU

فضای دیسک مورد نیاز: 363 MB

باز هم یک شاخه‌ی ساخت جدا ایجاد می‌کنیم:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Binutils را برای کامپایل شدن آماده می‌کنیم:

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
../binutils-2.22/configure --prefix=/tools \
--disable-nls --with-lib-path=/tools/lib
```

بسته را کامپایل می‌کنیم:

```
make
```

بسته را نصب می‌کنیم:

```
make install
```

حال لینکر را برای فاز «بازتنظیم» در فصل بعد آماده می‌کنیم:

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
```

۲-۴-۹) GCC - گذر دوم

زمان تقریبی ساخت: 7.0 SBU

فضای دیسک مورد نیاز: 1.5 GB

نسخه‌های پس از GCC 4.3 اجازه‌ی جست‌وجو برای پرنده‌های شروع را در محلّی که با prefix– مشخّص می‌شود را نمی‌دهند. از آن‌جا که دراین‌جا پرونده‌های شروعی که در tools/ قرار دارند برای ساخت یک کامپایلر کارای لینک‌شده به کتاب‌خانه‌های داخل tools/ ضروری هستند، وصله‌ی زیر برای بازگرداندن GCC به رفتار سابقش اعمال می‌شود:

```
patch -Np1 -i ../gcc-4.6.2-startfiles_fix-1.patch
```

در حالت استاندارد اسکریپت fixincludes برای تصحیح پرونده‌های سرآیند احتمالاً خراب شده اجرا می‌شود. از آن‌جا که هم‌اکنون GCC و Glibc نصب شده‌اند و پرونده‌های سرآیندشان نیازی به اصلاح ندارند، این اسکریپت موردنیاز نیست. در حقیقت اجرای این اسکریپت ممکن است محیط ساخت را با نصب سزآیندهای ثابت از سامانه‌ی میزبان به شاخه‌ی include خصوصی GCC آلوده سازد. با اجرای دستور زیر می‌توان مانع از اجرای اسکریپت fixincludes شد:

```
cp -v gcc/Makefile.in{,.orig}
sed 's@\./fixinc\.sh@-c true@' gcc/Makefile.in.orig > gcc/Makefile.in
```

برای ماشین‌های x86، یک ساخت خود راه‌انداز GCC از پرچم کامپایلر -fomit-frame-pointer استفاده می‌کند. ساخت‌های غیر خود راه‌انداز به صورت پیش‌فرض این پرچم را حذف می‌کنند، و هدف باید تولید کامپایلری باشد که اگر خود راه‌انداز بود دقیقاً به همین صورت می‌بود. دیتور sed زیر برای اجبار ساهت به استفاده از این پرچم به‌کار می‌رود:

```
cp -v gcc/Makefile.in{,.tmp}
sed 's/^T_CFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in.tmp \
> gcc/Makefile.in
```

دستور زیر مکان لینکر پویای پیش‌فرض GCC را برای استفاده از آنی که در tools/ نصب شده تغییر خواهد داد. این هم‌چنین usr/include/ را از مسیر جست‌وجوی include برای GCC برمی‌دارد. انجام دادن این کار اکنون به‌جای تنظیم پرونده‌ی مشخّصات پس از نصب، این اطمینان را می‌دهد که در طول ساخت واقعی GCC از لینکر پویای جدید استفاده می‌شود. این بدان معناست که تمامی دودویی‌هایی که در طول ساخت ایجاد می‌شوند به Glibc جدید لینک می‌شوند:

```
for file in \
$(find gcc/config -name linux64.h -o -name linux.h -o -name sysv4.h)
do
cp -uv $file{,.orig}
sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
-e 's@/usr@/tools@g' $file.orig > $file
echo '
#undef STANDARD_INCLUDE_DIR
#define STANDARD_INCLUDE_DIR 0
#define STANDARD_STARTFILE_PREFIX_1 ""
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
touch $file.orig
done
```

روی x86_64 بازنشاندن مشخصات multilib برای GCC این اطمینان را می‌دهد که تلاش به لینک کردن به کتاب‌خانه‌های روی میزبان نخواهد کرد:

```
case $(uname -m) in
x86_64)
for file in $(find gcc/config -name t-linux64) ; do \
cp -v $file{,.orig}
sed '/MULTILIB_OSDIRNAMES/d' $file.orig > $file
done
;;
esac
```

همانند نخستین ساخت، GCC نیازمند بسته‌های GMP، MPFR و MPC است:

```
tar -jxf ../mpfr-3.1.0.tar.bz2
mv -v mpfr-3.1.0 mpfr
tar -Jxf ../gmp-5.0.4.tar.xz
mv -v gmp-5.0.4 gmp
tar -zxf ../mpc-0.9.tar.gz
mv -v mpc-0.9 mpc
```

باز هم یک شاخه‌ی ساخت جدا ایجاد می‌شود:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

پیش از شروع ساخت GCC باید به خاطر داشت که هر متغیّر محلّی‌ای را که بر پرچم‌های بهینه‌سازی پیش‌فرض غلبه می‌کند، بازنشانی کرد.

برای آماده‌سازی GCC برای کامپایل:

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
../gcc-4.6.2/configure --prefix=/tools \
--with-local-prefix=/tools --enable-clocale=gnu \
--enable-shared --enable-threads=posix \
--enable-__cxa_atexit --enable-languages=c,c++ \
--disable-libstdcxx-pch --disable-multilib \
--disable-bootstrap --disable-libgomp \
--without-ppl --without-cloog \
--with-mpfr-include=$(pwd)/../gcc-4.6.2/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

در پایان یک پیوند نرم ایجاد می‌شود. بسیاری از برنامه‌ها و اسکریپت‌ها به جای gcc از cc استفاده می‌کنند، که برنامه‌ها را جامع و قابل استفاده در تمامی انواع سامانه‌های یونیکسی می‌کند که ممکن است همیشه در آن‌ها کامپایلر C گنو نصب نشده باشد. اجرای cc مدیر سامانه را برای انتخاب در مورد این که کدام کامپایلر را نصب کند آزاد می‌گذارد:

```
ln -vs gcc /tools/bin/cc
```

Tcl (۱۰-۴-۲

زمان تقریبی ساخت: 1.1 SBU

فضای دیسک مورد نیاز: 363 MB

این بسته و سه تای بعدی (Expect، DejaGNU و Check) برای پشتیبانی مجموعه‌های آزمایشی GCC و Binutils و دیگر بسته‌ها نصب می‌شوند. ممکن است نصب چهار بسته برای مقاصد

آزمایشی زیاده‌روی به‌نظر برسد، ولی اگر ضروری نباشد هم قوّت قلب بسیاری جهت اطمینان از این که ابزارهای مهم به‌خوبی کار می‌کنند به همراه دارد. حتا اگر بسته‌های آزمایشی در این فصل اجرا نشوند (اجباری نیستند)، این بسته‌ها برای اجرای بسته‌های آزمایشی در فصل سوم مورد نیازند.

Tcl را برای کامپایل شدن آماده می‌کنیم:

```
cd unix
./configure --prefix=/tools
```

بسته را می‌سازیم:

```
make
```

اکنون کامپایل کامل شده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌های ازمایشی برای ابزارهای موقّتی این فصل اجباری نیست. به‌هرحال برای اجرای بسته‌ی آزمایشی Tcl باید دستور زیر را اجرا کرد:

```
TZ=UTC make test
```

برای نصب بسته:

```
make install
```

برای این‌که علائم خطایابی بتوانند بعداً برداشته شوند، باید کتاب‌خانه‌ی نصب شده قابل نوشتن شود:

```
chmod -v u+w /tools/lib/libtcl8.5.so
```

بسته‌ی بعدی، Expect برای ساخت نیاز به سرآیندهای Tcl دارد که با دستور زیر نصب می‌شوند:

```
make install-private-headers
```

حال می‌بایست یک پیوند نرم مهم را ایجاد کرد:

```
ln -sv tclsh8.5 /tools/bin/tclsh
```

محتویات Tcl

برنامه‌های نصب شده: tclsh(پیوند به tclsh8.5) و tclsh8.5

کتاب‌خانه‌های نصب شده: libtcl8.5.so و libtclstub8.5.a

Expect (۱۱-۴-۲

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 4.1 MB

نخست اسکریپت پیکربندی Expect را مجبور می‌کنیم تا به‌جای /usr/local/bin/stty که ممکن است روی سامانه‌ی میزبان یافت شود، از /bin/stty استفاده کند. این کار این اطمینان را می‌دهد که ابزار بسته‌ی آزمایشی برای ساخت‌های پایانی زنجیرابزار ما سالم می‌ماند:

```
cp -v configure{,.orig}
sed 's:/usr/local/bin:/bin:' configure.orig > configure
```

حال آن را برای کامپایل شدن آماده می‌کنیم:

```
./configure --prefix=/tools --with-tcl=/tools/lib \
--with-tclinclude=/tools/include
```

برای ساخت بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Expect، باید دستور زیر را وارد کرد:

```
make test
```

برای نصب بسته:

```
make SCRIPTS="" install
```

محتویات Expect

برنامه‌های نصب شده: expect

کتاب‌خانه‌های نصب شده: libexpect-5.45.a

۲-۴-۱۲) DejaGNU

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 6.1 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

ساخت و نصب بسته:

```
make install
```

برای بررسی نتیجه:

```
make check
```

محتویات DejaGNU

برنامه‌های نصب شده: runtest

۲-۴-۱۳) Check

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 4.8 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

ساخت بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Check، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

محتویات Check

کتاب‌خانه‌های نصب شده: libcheck.{a,so}

۲-۴-۱۴) Ncurses

زمان تقریبی ساخت: 0.7 SBU

فضای دیسک مورد نیاز: 30 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools --with-shared \
--without-debug --without-ada --enable-overwrite
```

کامپایل بسته:

```
make
```

این بسته یک مجموعه‌ی آزمایشی دارد، ولی تنها پس از نصب بسته قابل اجرا است. بررسی‌ها در شاخه‌ی /test قرار می‌گیرند.

براین نصب بسته:

```
make install
```

۲-۴-۱۵) Bash

زمان تقریبی ساخت: 0.5 SBU

فضای دیسک مورد نیاز: 35 MB

نخست می‌بایست وصله‌ی زیر را برای تصحیح خطاهای بالادستی اعمال می‌کنیم:

```
patch -Np1 -i ../bash-4.2-fixes-4.patch
```

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools --without-bash-malloc
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Bash، باید دستور زیر را وارد کرد:

```
make tests
```

برای نصب بسته:

```
make install
```

برای برنامه‌هایی که از sh به عنوان پوسته استفاده می‌کنند یک پیوند نرم ایجاد می‌کنیم:

```
ln -vs bash /tools/bin/sh
```

۲-۴-۱۶) Bzip2

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 4.8 MB

بسته‌ی Bzip2 شامل اسکریپت پیکربندی نیست و با این دستور کامپایل و بررسی می‌شود:

```
make
```

برای نصب بسته:

```
make PREFIX=/tools install
```

۲-۴-۱۷) Coreutils

زمان تقریبی ساخت: 0.7 SBU

فضای دیسک مورد نیاز: 88 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools --enable-install-program=hostname
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Bash، باید دستور زیر را وارد کرد:

```
make RUN_EXPENSIVE_TESTS=yes check
```

برای نصب بسته:

```
make install
```

دستور بالا su را نصب نمی‌کند، زیرا نمی‌توان به عنوان یک کاربر غیرممتاز برنامه را با دسترسی ریشه اجرا کرد. به وسیله‌ی نصب دستی آن با نامی دیگر می‌توانیم از آن برای اجرای بررسی‌ها در سامانه‌ی نهایی به عنوان یک کاربر غیرممتاز استفاده کنیم:

```
cp -v src/su /tools/bin/su-tools
```

۱۸-۴-۲) Diffutils

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 6.1 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Diffutils، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۱۹) File

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 9.5 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی File، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۰) Findutils

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 20 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Findutils، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۱) Gawk

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 28 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Gawk، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۲) Gettext

زمان تقریبی ساخت: 0.8 SBU

فضای دیسک مورد نیاز: 82 MB

برای مجموعه ابزارهای موقّتی، تنها نیاز است که یک دودویی از Gettext ساخته و نصب شود.

آماده‌سازی برای کامپایل شدن:

```
cd gettext-tools
./configure --prefix=/tools --disable-shared
```

کامپایل بسته:

```
make -C gnulib-lib
make -C src msgfmt
```

از آن‌جا که تنها یک دودویی کامپایل شده، امکان اجرای مجموعه‌ی آزمایشی بدون کامپایل کردن کتاب‌خانه‌های پشتیبانی اضافی از بسته‌ی Gettext نیست. پس تلاش برای اجرای مجموعه‌ی ازمایشی در این مرحله توصیه نمی‌شود. برای نصب دودویی msgfmt:

```
cp -v src/msgfmt /tools/bin
```

۲-۴-۲۳) Grep

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 18 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools \
--disable-perl-regexp
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Grep، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۴) Gzip

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 3.3 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Grep، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۵) M4

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 11.6 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی M4، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۶) Make

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 9.6 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Make، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۷) Patch

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 1.9 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Make، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۲۸) Perl

زمان تقریبی ساخت: 1.8 SBU

فضای دیسک مورد نیاز: 223 MB

نخست باید وصله‌ی زیر را برای وفق دادن برخی مسیرها با کتاب‌خانه‌ی C اعمال کرد:

```
patch -Np1 -i ../perl-5.14.2-libc-1.patch
```

آماده‌سازی برای کامپایل شدن:

```
sh Configure -des -Dprefix=/tools
```

کامپایل بسته:

```
make
```

با این‌که پرل با یک مجموعه‌ی آزمایشی همراه است، بهتر است که تا نصب آن در فصل بعدی صبر کرد. در حال حاضر تنها نیاز است تا چند تا از ابزارهای سیستمی و کتاب‌خانه‌ها نصب شوند:

```
cp -v perl cpan/podlators/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.14.2
cp -Rv lib/* /tools/lib/perl5/5.14.2
```

۲-۴-۲۹) Sed

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 8.0 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Sed، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

Tar (۳۰-۴-۲

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 20.9 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Tar، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

Texinfo (۳۱-۴-۲

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 20 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Texinfo، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

Xz (۳۲-۴-۲

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 14 MB

آماده‌سازی برای کامپایل شدن:

```
./configure --prefix=/tools
```

کامپایل بسته:

```
make
```

اکنون کامپایل به پایان رسیده است. همان‌گونه که پیش‌تر بحث شد، اجرای بسته‌ی آزمایشی برای ابزارهای موقّتی در این فصل لازم نیست. به هر حال برای اجرای بسته‌ی آزمایشی Xz، باید دستور زیر را وارد کرد:

```
make check
```

برای نصب بسته:

```
make install
```

۲-۴-۳۳) تُنُک‌سازی

مراحل این قسمت اختیاری هستند، ولی اگر پارتیشن سیستم‌عامل جدیدنسبتاً کوچک باشد، مفید است بدانیم که اقلام غیرضروری می‌توانند برداشته شوند. پرونده‌های اجرایی و کتاب‌خانه‌هایی که تا به حال ساخته شده‌اند حدود ۷۰ مگابایت از علائم اشکال‌زدایی غیرلازم به همراه دارند. با این دستور می‌توان آن علائم را برداشت:

```
strip --strip-debug /tools/lib/*
strip --strip-unneeded /tools/{,s}bin/*
```

برای ذخیره‌ی بیش‌تر، می‌توان مستندات را نیز برداشت:

```
rm -rf /tools/{,share}/{info,man,doc}
```

با این کار حداقل ۸۵۰ مگابایت فضای خالی در $LFS وجود دارد که می‌تواند برای ساخت و نصب Glibc در فاز بعدی استفاده شود. اگر بتوان Glibc را ساخت و نصب کرد، می‌توان بقیه‌ی سیستم‌عامل را نیز ساخته و نصب نمود.

۲-۴-۳۴) تغییر مالکیت

نکته: دستوراتی که در ادامه می‌آیند باید هنگامی که به جای کاربر lfs به عنوان کاربر ریشه وارد سیستم شده‌ایم اجرا شود. هم‌چنین باید بررسی شود که $LFS در محیط ریشه هم تنظیم شده باشد.

در حال حاضر شاخه‌ی $LFS/tools در مالکیت کاربر lfs، کاربری که تنها روی سامانه‌ی میزبان وجود دارد است. اگر این شاخه همین‌گونه که هست نگه داشته شود، پرونده‌ها در مالکیت شناسه‌ی کاربری‌ای بدون حساب متناظر خواهند بود. این خطرناک است زیرا که ممکن است یک حساب

کاربری که بعدها ساخته می‌شود این شناسه را تصاحب کند و مالک این شاخه و تمام پرونده‌های درون آن شود.

برای اجتناب از این امر می‌توان کاربر lfs را بعداً هنگام ساخت پرونده‌ی /etc/passwd به سیستم‌عامل جدیداضافه کرد و مواظب بود که همان شناسه‌ی کاربری و گروهی روی سامانه‌ی میزبان را تصاحب کند. اما از آن بهتر می‌توان مالکیت شاخه‌ی $LFS/tools را با دستور زیر به کاربر ریشه واگذار کرد:

```
chown -R root:root $LFS/tools
```

اگر چه این شاخه می‌تواند به محض اتمام سیستم‌عامل جدیدپاک شود، می‌تواند برای ساخت سیستم‌های عامل دیگری از همین نوع با امکانات بیش‌تر نگه‌داشته شود.

فصل سوم

ساخت سیستم‌عامل جدید

۳-۱) نصب نرم‌افزار سیستمی پایه

در این فصل ما وارد مرحله‌ی ساخت می‌شویم و ایجاد سیستم‌عامل جدیدرا با جدّیت در پیش می‌گیریم. برای این‌کار به سامانه‌ی گنو/لینوکسی موقّتی chroot می‌کنیم، چند آماده‌سازی نهایی را لنجام می‌دهیم و سپس شروع به نصب بسته‌ها می‌کنیم.

۳-۱-۱) آماده‌سازی فایل‌سیستم هسته‌ی مجازی

قایل‌سیستم‌های متنوعی که توسّط هسته صادر می‌شوند برای ارتباط برقرار کردن با خود هسته به کار می‌روند. این فایل‌سیستم‌ها مجازی هستند که برای آن‌ها هیچ فضای دیسکی استفاده نمی‌شود. محتویات فایل‌سیستم در حافظه قرار می‌گیرند.

برای شروع شاخه‌هایی را که فایل‌سیستم روی آن‌ها سوار می‌شود ایجاد می‌کنیم:

```
mkdir -v $LFS/{dev,proc,sys}
```

ایجاد گره‌های دستگاه ابتدایی

هنگامی که هسته سامانه را راه‌اندازی می‌کند، نیاز به وجود چند گره دستگاه دارد، به طور ویژه دستگاه‌های console و null. گره‌های دستگاه باید روی دیسک سخت ایجاد شوند تا پیش از شروع udevd و همین‌طور هنگامی که لینوکس با init=/bin/bash شروع می‌شود در دست‌رس باشند. این دستگاه‌ها با دستور زیر ایجاد می‌شوند:

```
mknod -m 600 $LFS/dev/console c 5 1
mknod -m 666 $LFS/dev/null c 1 3
```

سوارکردن و مسکون کردن dev/

روش پیشنهادی مسکون کردن شاخه‌ی dev/ با دستگاه‌ها، سوار کردن یک فایل‌سیستم مجازی (مانند tmpfs) روی این شاخه و اجازه دادن به دستگاه‌ها برای ساخته‌شدن پویا روی فایل‌سیستم یه محض تشخیص آن‌ها یا دست‌رسی به آن‌ها است. تولید دستگاه معمولاً در طول فرایند راه‌اندازی و توسّط Udev انجام می‌پذیرد. از آن‌جا که این سامانه‌ی حدید هنوز Udev ندارد و هنور راه‌اندازی نشده است، لازم است dev/ را به صورت دستی سوار و مسکون کنیم. این کار از طریق سوار کردن متّصل شاخه‌ی dev/ میزبان صورت می‌گیرد. سوار کردن متّصل توع خاصّی از سوار کردن است که اجازه می‌دهد از شاخه یا نقطه‌ی اتّصال یا مکان دیگری یک آیینه ایجاد شود. برای نیل به این هدف از دستور زیر استفاده می‌شود:

```
mount -v --bind /dev $LFS/dev
```

سوار کردن فایل‌سیستم هسته‌ی مجازی

حال می‌بایست فایل‌سیستم‌های هسته‌ی مجازی باقی‌مانده را سوار کرد:

```
mount -vt devpts devpts $LFS/dev/pts
mount -vt tmpfs shm $LFS/dev/shm
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
```

۳-۱-۲) وارد شدن به محیط chroot

خال زمان ورود به محیط chroot برای شروع ساخت و نصب سیستم‌عامل نهایی است. به عنوان کاربر ریشه باید دستورات زیر اجرا شوند تا وارد قلمرویی شویم که در حال حاضر تنها با ابزارهای موقّتی مسکون شده‌است:

```
chroot "$LFS" /tools/bin/env -i \
HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
/tools/bin/bash --login +h
```

باید توجّه داشت که در اعلان bash نوشته خواهد شد «I have no name!». این امر طبیعی است، زیرا که پروندهی /etc/passwd هنوز ساخته نشده است.

۳-۱-۳) ایجاد شاخه‌ها

حال وقت آن است که برخی ساختارها را در سیستم‌عامل جدیدایجاد کنیم. با دستورات زیر یک درخت شاخه‌ی استاندارد ایجاد می‌شود:

```
mkdir -pv /{bin,boot,etc/{opt,sysconfig},home,lib,mnt,opt,run}
mkdir -pv /{media/{floppy,cdrom},sbin,srv,var}
install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{doc,info,locale,man}
mkdir -v /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
for dir in /usr /usr/local; do
ln -sv share/{man,doc,info} $dir
done
case $(uname -m) in
x86_64) ln -sv lib /lib64 && ln -sv lib /usr/lib64 ;;
esac
mkdir -v /var/{log,mail,spool}
ln -sv /run /var/run
ln -sv /run/lock /var/lock
mkdir -pv /var/{opt,cache,lib/{misc,locate},local}
```

شاخه‌ها به صورت پیش‌فرض با سطح دست‌رسی ۷۵۵ ایجاد می‌شوند، ولی این برای همه‌ی شاخه‌ها مطلوب نیست. در دستورات بالا دو تغییر داده شده است، یکی برای شاخه‌ی خانگی کاربر ریشه و دیگری برای شاخه‌های پرونده‌های موقّتی.

تغییر دست‌رسی نخست این اطمینان را می‌دهد که هرکسی نمی‌تواند وارد شاخه‌ی root/ شود. دومین تغییر این اطمینان را می‌دهد که همه‌ی کاربران می‌توانند در شاخه‌های tmp و /var/tmp بنویسند، ولی نمی‌توانند پرونده‌های کاربران دیگر را از آن‌ها بردارند.

۳-۱-۴) ایجاد پرونده‌ها و پیوندهای نرم ضروری

برخی برنامه‌ها از مسیرهای سخت به برنامه‌هایی که هنوز وجود ندارند استفاده می‌کنند. برای ارضای این برنامه‌ها چند پیوند نرم می‌سازیم که با پرونده‌های واقعی جایگزین شوند:

```
ln -sv /tools/bin/{bash,cat,echo,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv /tools/lib/libstdc++.so{,.6} /usr/lib
sed 's/tools/usr/' /tools/lib/libstdc++.la > /usr/lib/libstdc++.la
ln -sv bash /bin/sh
```

یک سامانه‌ی گنو/لینوکسی سالم سیاهه‌ای از فایل‌سیستم‌های سوارشده را در پرونده‌ی /etc/mtab نگه‌داری می‌کند. در حالت عادی این پرونده وقتی فایل‌سیستم جدیدی را سوار می‌کنیم ایجاد می‌شود. از آن‌جایی که در محیط chrootمان هیچ فایل‌سیستمی را سوار نخواهیم کرد، باید پرونده‌ای خالی برای تسهیلاتی که انتظار وجود این پرونده را دارند ایجاد کرد:

```
touch /etc/mtab
```

برای این که کاربر ریشه قادر به ورود باشد و نام root شناخته شود، باید ورودی‌های متناسب در پرونده‌های /etc/passwd و /etc/group ثبت شوند.

پرونده‌ی /etc/passwd با دستور زیر ایجاد می‌شود:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/dev/null:/bin/false
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
EOF
```

گذرواژه‌ی واقعی برای root بعداً تنظیم می‌شود.

پرونده‌ی /etc/passwd نیز با اجرای دستور زیر ایجاد می‌شود:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
```

```
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
mail:x:34:
nogroup:x:99:
EOF
```

برای اصلاح اعلان «I have no name!» باید پوسته‌ی جدیدی باز کرد. از آن‌جا که یک Glibc کامل در فصل پیش نصب شد و پرونده‌های /etc/passwd/ و /etc/group/ ساخته شده‌اند، تفکیک نام کاربری و نام گروه اکنون کار می‌کند:

```
exec /tools/bin/bash --login +h
```

برنامه‌های login، agetty و init به همراه چند برنامه‌ی دیگر از تعدادی پرونده‌ی گزارش برای ضبط اطّلاعاتی نظیر این که چه کسی چه زمانی وارد سامانه شداستفاده می‌کنند. با این حال اگر این پرونده از قبل موجود نباشند، این اطّلاعات جایی نوشته نخواهند شد. با دستورات زیر می‌توان پرونده‌های گزارش را ایجاد کرد و به آن‌ها دست‌رسی مناسب را اعطا کرد:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chgrp -v utmp /var/run/utmp /var/log/lastlog
chmod -v 664 /var/run/utmp /var/log/lastlog
chmod -v 600 /var/log/btmp
```

۳-۱-۵) سرآیندهای API لینوکس

زمان تقریبی ساخت: 1 SBU

فضای دیسک مورد نیاز: 515 MB

هسته‌ی لینوکس باید برای استفاده‌ی کتاب‌خانه‌ی C سامانه (در این‌جا Glibc) یک رابط برنامه‌نویسی نرم‌افزار فراهم کند.این کار با نصب سرآیندهای C مختلفی که همراه با تربال منبع هسته‌ی لینوکس ارائه می‌شوند امکان‌پذیر است.

ابتدا باید مطمئن شد پرونده و پیش‌نیازی از فعّالیت‌های پیشین باقی نمانده است:

```
make mrproper
```

حال سرآیندهای هسته‌ی قابل مشاهده برای کاربر را از منبع بررسی کرده و از حالت فشرده خارج می‌کنیم. آن‌ها در یک شاخه‌ی محلّی میانی قرار گرفته‌اند و در مکان مورد نیاز رونوشت شده‌اند، زیرا که فرایند استخراج هر پرونده‌ی موجودی در شاخه‌ی مقصد را برمی‌دارد. هم‌چنین چند پرونده‌ی مخفی وجود دارد که توسّط توسعه‌دهندگان هسته استفاده می‌شوند و از شاخه‌ی میانی برداشته می‌شوند.

```
make headers_check
make INSTALL_HDR_PATH=dest headers_install
find dest/include \( -name .install -o -name ..install.cmd \) -delete
cp -rv dest/include/* /usr/include
```

Man-pages (۶-۱-۳

زمان تقریبی ساخت: کم‌تر از 1 SBU

فضای دیسک مورد نیاز: 21 MB

با این دستور نصب می‌شود:

```
make install
```

Glibc (۷-۱-۳

زمان تقریبی ساخت: 14.2 SBU

فضای دیسک مورد نیاز: 856 MB

سامانه‌ی ساخت Glibc خودشمول است و کاملاً درست نصب می‌شود، ولو این‌که پرونده‌ی مشخّصات کامپایلر و لینکر هنوز به tools/ اشاره می‌کنند. مشخصات و لینکر نمی‌توانند پیش از نصب Glibc تنظیم شوند، زیرا بررسی‌های پیکربندی خودکار Globc نتایج نادرست خواهند داد و هدف دستیابی به یک ساخت بی‌نقص مغلوب می‌شود.

هنگام اجرای make install اسکریپتی به نام test-installation.pl یک بررسی سلامت کوچک

روی Glibc تازه نصب‌شده انجام می‌دهد. به هرحال، چون زنجیرابزار هنوز به شاخه‌ی tools/ اشاره می‌کند، تست سلامت روی Glibc اشتباهی انجام می‌شود. بدین صورت می‌توان اسکریپت را وادار کرد تا Glibcای را که تازه نصب کرده‌ایم بررسی کند:

```
DL=$(readelf -l /bin/sh | sed -n 's@.*interpret.*/tools\(.*\)]$@\1@p')
sed -i "s|libs -o|libs -L/usr/lib -Wl,-dynamic-linker=$DL -o|" \
scripts/test-installation.pl
unset DL
```

به علاوه در اسکریپت فوق باگی وجود دارد که تلاش می‌کند یک برنامه‌ی بررسی را به کتاب‌خانه‌ای که با make install نصب نشده است پیوند دهد. دستور زیر این مشکل را برطرف می‌کند:

```
sed -i -e 's/"db1"/& \&\& $name ne "nss_test1"/' scripts/test-installation.pl
```

اسکریپت پوسته‌ای ldd شامل ترکیبی مخصوص Bash است. اگر پوسته‌ی دیگری هم نصب باشد می‌توانبرای حصول اطمینان از درست کار کردن این اسکریپت آن را با دستور زیر تغییر داد:

```
sed -i 's|@BASH@|/bin/bash|' elf/ldd.bash.in
```

دو باگ در Glibc که ممکن است منجر به خرابی شود را برطرف می‌کنیم:

```
patch -Np1 -i ../glibc-2.14.1-fixes-1.patch
patch -Np1 -i ../glibc-2.14.1-sort-1.patch
```

باگی که Glibc را از کامپایل شدن با GCC-4.6.2 باز می‌دارد را نیز برطرف می‌کنیم:

```
patch -Np1 -i ../glibc-2.14.1-gcc_fix-1.patch
```

یک عدم تعادل پشته را که در برخی شرایط رخ می‌دهد را برطرف می‌کنیم:

```
sed -i '195,213 s/PRIVATE_FUTEX/FUTEX_CLOCK_REALTIME/' \
nptl/sysdeps/unix/sysv/linux/x86_64/pthread_rwlock_timed{rd,wr}lock.S
```

مستندات Glibc ساخت Glibc را در یک شاخه‌ی ساخت تخصیص داده شده در بیرون شاخه‌ی منبع توصیه می‌کنند:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

همانند فصل گذشته باید برای ماشین‌های x86 پرچم‌های مورد نیاز را به CFLAGS افزود. این‌جا هم‌چنین بهینه‌سازی کتاب‌خانه برای کامپایلر gcc به منظور افزایش سرعت کامپایل و بازدهی بسته‌ها تنظیم شده است.

```
case `uname -m` in
i?86) echo "CFLAGS += -march=i486 -mtune=native -O3 -pipe" > configparms ;;
esac
```

آماده سازی برای کامپایل:

```
../glibc-2.14.1/configure --prefix=/usr \
--disable-profile --enable-add-ons \
--enable-kernel=2.6.25 --libexecdir=/usr/lib/glibc
```

کامپایل بسته:

```
make
```

پیش از اجرای بررسی‌ها، پرونده‌ای را از درخت منبع به درخت ساخت خود رونوشت می‌کنیم تا از دو خطای بررسی جلوگیری شود، سپس نتیجه را بررسی می‌کنیم:

```
cp -v ../glibc-2.14.1/iconvdata/gconv-modules iconvdata
make -k check 2>&1 | tee glibc-check-log
grep Error glibc-check-log
```

احتمالاً در این‌جا یک خطای مورد انتظار (نادیده گرفته شده) رخ خواهد داد. با این‌که این پیام بی‌ضرر است، سکوی نصب Glibc شکایت از فقدان /etc/ld.so.conf می‌کند. با این دستور می‌توان از این هشدار جلوگیری کرد:

```
touch /etc/ld.so.conf
```

نصب بسته:

```
make install
```

می‌توان در این‌جا سرآیندهای مرتبط با NIS و RPC را که به صورت پیش‌فرض نصب نمی‌شوند، نصب کرد:

```
cp -v ../glibc-2.14.1/sunrpc/rpc/*.h /usr/include/rpc
cp -v ../glibc-2.14.1/sunrpc/rpcsvc/*.h /usr/include/rpcsvc
cp -v ../glibc-2.14.1/nis/rpcsvc/*.h /usr/include/rpcsvc
```

منطقه‌ها که می‌توانند سامانه را به زبان‌های دیگر پاسخ‌گو کنند با دستورات بالا نصب نمی‌شوند. هیچ کدام از منطقه‌ها ضروری نیستند، ولی اگر برخی از آن‌ها نباشند، مجموعه‌های آزمایشی بسته‌های بعدی موردهای بررسی مهمی را از قلم خواهند انداخت.

منطقه‌ها می‌توانند به صورت جدا از هم با برنامه‌ی localedef نصب شوند. مثلا نخستین دستور localedef زیر تعریف منطقه‌ی مستقل از مجموعه نویسه‌ی /usr/share/i18n/locales/cs_CZ را با تعریف نقشه‌ی نویسه‌ی /usr/share/i18n/charmaps/UTF-8.gz ترکیب کرده و نتیجه را پرونده‌ی /usr/lib/locale/locale-archive الحاق می‌کند. دستورات زیر کم‌ترین مجموعه منطقه‌های را برای پوشش بهینه‌ی بررسی‌ها نصب می‌کند:

```
mkdir -pv /usr/lib/locale
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i en_US -f UTF-8 en_US.UTF-8
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
localedef i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
```

به جای این عمل می‌توان تمام منطقه‌هایی که در پرونده‌ی glibc-2.14.1/localedata/SUPPORTED فهرست شده‌اند را یک‌جا با دستور وقت‌گیر زیر نصب کرد:

```
make localedata/install-locales
```

پیکربندی Glibc

لازم است پرونده‌ی /etc/nsswitch.conf ایجاد شود، زیرا با این که Glibc هنگام غیاب یا خرابی این پرونده پیش‌نیازها را فراهم می‌کند، پیش‌نیازهایش در یک محیط شبکه‌ای خوب کار نمی‌کنند. هم‌چنین موقعیت زمانی باید پیکربندی شود.

با اجرای دستور زیر یک پرونده‌ی جدید /etc/nsswitch.conf ایجاد می شود:

```
cat > /etc/nsswitch.conf << "EOF"
# Begin /etc/nsswitch.conf
```

```
passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

# End /etc/nsswitch.conf
EOF
```

یکی از راه‌های تعیین منطقه‌ی زمانی محلّی اجرای اسکریپت زیر است:

```
tzselect
```

پس از پاسخ به چند پرسش درباره‌ی مکان، اسکریپت نام منطقه‌ی زمانی را برون‌دهی می‌کند (برای مثال Asia/Tehran).

سپس پرونده‌ی etc/localtime/ را با دستور زیر ایجاد می‌کنیم:

```
cp -v --remove-destination /usr/share/zoneinfo/<xxx> \
/etc/localtime
```

که در آن <xxx> با نام منطقه‌ی زمانی انتخاب شده تعویض می‌شود.

پیکربندی بارکننده‌ی پویا

به صورت پیش‌فرض بارکننده‌ی پویا (lib.ld=linux.so.2.) برای کتاب‌خانه‌های پویایی که برای برنامه‌های درحال اجرا مورد نیاز هستند، در lib/ و usr/lib/ جست‌وجو می‌کند. با این حال اگر کتاب‌خانه‌هایی در شاخه‌هایی غیر از آن دو باشند، نیاز است به پرونده‌ی etc/ld.so.conf/ افزوده شوند تا توسّط بارکننده‌ی پویا یافته شوند. دو شاخه که معمولاً به داشتن کتاب‌خانه‌های اضافی شهره اند usr/local/lib/ و opt/lib/ هستند، پس آن‌ها را به مسیر جست‌وجوی بارکننده‌ی پویا می‌افزاییم:

```
cat > /etc/ld.so.conf << "EOF"
# Begin /etc/ld.so.conf
```

```
/usr/local/lib
/opt/lib
EOF
```

در صورت تمایل، بارکننده‌ی پویا می‌تواند هم‌چنین شاخه‌ای را جست‌وجو کند و محتویات پرونده‌های یافته‌شده در آن‌جا را شامل شود. عموماً پرونده‌های موجود در این شاخه یک خط هستند که مسیر کتاب‌خانه‌ی دل‌خواه را مشخّص می‌کنند. برای افزودن این قابلیت از دستور زیر استفاده می‌شود:

```
cat >> /etc/ld.so.conf << "EOF"
# Add an include directory
include /etc/ld.so.conf.d/*.conf
EOF
mkdir /etc/ld.so.conf.d
```

۳-۱-۸) بازتنظیم زنجیرابزار

حال که کتاب‌خانه‌های C نهایی نصب شدند، وقت آن است که دوباره زنجیرابزار را تنظیم کنیم. زنجیرابزار به گونه‌ای تنظیم خواهد شد که هر برنامه‌ی تازه کامپایل شده‌ای را به این کتاب‌خانه‌های جدید پیوند دهد. این شبیه فرایندی است که در فصل پیش انجام شد، ولی با تغییرات معکوس.

نخست می‌بایست از لینکر tools/ پشتیبان گرفت و آن را با لینکر تنظیم‌شده‌ای که قبلاً ساختیم جایگزین کرد. هم‌چنین باید پیوندی به همتای آن در tools/$(gcc -dumpmachine)/bin/ ایجاد کرد:

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

سپس باید پرونده‌ی مشخّصات GCC را بهبود مبخشید تا به لینکر پویای جدید اشاره کند. به طور ساده پاک کردن همه‌ی نمونه‌های «/tools» می‌بایست مسیر درست به لینکر پویا را حاصل دهد. هم‌چنین باید تنظیم پرونده‌ی مشخّصات را به گونه‌ای تغییر داد که GCC بداند کجا پرونده‌های شروع و سرآیندهای صحیح Glibc را بیابد. یک دستور sed این را انجام می‌دهد:

```
gcc -dumpspecs | sed -e 's@/tools@@g' \
-e '/\*startfile_prefix_spec:/{n;s@.*@/usr/lib/ @}' \
-e '/\*cpp:/{n;s@$@ -isystem /usr/include@}' > \
`dirname $(gcc --print-libgcc-file-name)`/specs
```

در این‌جا لازم است مطمئن شویم که توابع پایه‌ای کامپایل و لینک زنجیرابزار تنظیم‌شده، آن‌گونه که انتظار می‌رود کار می‌کنند. برای این‌کار می‌بایست بررسی‌های زیر انجام شوند:

```
echo 'main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

اگر همه‌چیز درست کار کند، نباید خطایی وجود داشته باشد و برونده آخرین دستور مطابق زیر خواهد بود:

```
/usr/lib/crt1.o succeeded
/usr/lib/crti.o succeeded
/usr/lib/crtn.o succeeded
```

باید بررسی کرد که کامپایلر به دنبال پرونده‌های سرآیند درست می‌گردد:

```
grep -B1 '^ /usr/include' dummy.log
```

این دستور باید با موفّقیت با برونده زیر انجام شود:

```
#include <...> search starts here:
/usr/include
```

سپس باید بررسی شود که لینکر جدید با مسیرهای جست‌وجوی صحیح استفاده می‌شود:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
SEARCH_DIR("/tools/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib");
```

بعد باید اظمینان پیدا کنیم که داریم از کتاب‌خانه‌ی C درست استفاده می‌کنیم:

```
grep "/lib.*/libc.so.6 " dummy.log
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
attempt to open /lib/libc.so.6 succeeded
```

در پایان باید از استفاده‌ی GCC از لینکر پویای صحیح اطمینان پیدا کرد:

```
grep found dummy.log
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برون‌ده آخرین دستور به شکل زیر خواهد بود:

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

پس از این که همه‌چیز درست کار کرد، پرونده‌های آزمایشی را تمیز میی کنیم:

```
rm -v dummy.c a.out dummy.log
```

۳-۱-۹) Zlib

زمان تقریبی ساخت: کم‌تر از 1 SBU

فضای دیسک مورد نیاز: 2.8 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

کتاب‌خانه‌ی به‌اشتراک گذاشته شده نیاز دارد به lib/ منتقل شود و درنتیجه پرونده‌ی so. در usr/lib/ نیاز خواهد داشت که دوباره ایجاد شود:

```
mv -v /usr/lib/libz.so.* /lib
ln -sfv ../../lib/libz.so.1.2.6 /usr/lib/libz.so
```

۳-۱-۱۰) File

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 9.5 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۱۱) Binutils

زمان تقریبی ساخت: 1.9 SBU

فضای دیسک مورد نیاز: 307 MB

با دستور زیر بررسی می‌شود که PTYها به خوبی درون محیط chroot کار می‌کنند یا نه:

```
expect -c "spawn ls"
```

این دستور باید برون‌ده زیر را داشته باشد:

```
spawn ls
```

از آن‌جا که در دستورات پیکربندی خودکار یک پرونده‌ی جدید standards.info نصب می‌شود، باید از نصب پرونده‌ی منقضی شده‌ی آن جلوگیری کرد:

```
rm -fv etc/standards.info
sed -i.bak '/^INFO/s/standards.info //' etc/Makefile.in
```

دو بررسی که هنگام استفاده از GCC-4.6.2 خطا می‌دهند را تصحیح می‌کنیم:

```
sed -i "/exception_defines.h/d" ld/testsuite/ld-elf/new.cc
sed -i "s/-fvtable-gc //" ld/testsuite/ld-selective/selective.exp
```

مستندات این بسته ساخت آن را در یک شاخه‌ی ساخت تخصیص داده شده در بیرون شاخه‌ی منبع توصیه می‌کنند:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

آماده سازی برای کامپایل:

```
../binutils-2.22/configure --prefix=/usr --enable-shared
```

کامپایل بسته:

```
make tooldir=/usr
```

بررسی نتیجه:

```
make -k check
```

نصب بسته:

```
make tooldir=/usr install
```

نصب سرآیند libiberty که توسط برخی بسته‌ها مورد نیاز است:

```
cp -v ../binutils-2.22/include/libiberty.h /usr/include
```

GMP (۱۲-۱-۳)

زمان تقریبی ساخت: 1.9 SBU

فضای دیسک مورد نیاز: 307 MB

آماده سازی برای کامپایل:

```
./configure --prefix=/usr --enable-cxx --enable-mpbsd
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check 2>&1 | tee gmp-check-log
```

باید اظمینان حاصل کرد که همه‌ی ۱۶۴ آزمایش در مجموعه‌ی آزمایشی قبول می‌شوند. نتیجه را می‌توان با دستور زیر بررسی کرد:

```
awk '/tests passed/{total+=$2} ; END{print total}' gmp-check-log
```

نصب بسته:

```
make install
```

در صورت تمایل می‌توان مستندات را نیز نصب کرد:

```
mkdir -v /usr/share/doc/gmp-5.0.4
cp -v doc/{isa_abi_headache,configuration} doc/*.html \
/usr/share/doc/gmp-5.0.4
```

MPFR (۱۳-۱-۳)

زمان تقریبی ساخت: 1.1 SBU

فضای دیسک مورد نیاز: 272.1 MB

اعمال وصله‌ای که تعدادی از باگ‌ها را در نسخه‌ی ۳.۱ برطرف می‌کند:

```
patch -Np1 -i ../mpfr-3.1.0-fixes-1.patch
```

آماده سازی برای کامپایل:

```
./configure --prefix=/usr --enable-thread-safe \
--docdir=/usr/share/doc/mpfr-3.1.0
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

نصب مستندات:

```
make html
make install-html
```

MPC (۱۴-۱-۳

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 10.5 MB

آماده سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

GCC (۱۵-۱-۳

زمان تقریبی ساخت: 47 SBU

فضای دیسک مورد نیاز: 1.7 MB

ابتدا باید برای پرهیز از نصب libibertyیک عمل جانشینی انجام داد تا به‌جای آن، نسخه‌ی libiberty فراهم‌شده توسّط Binutils نصب شود:

```
sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in
```

مانند گذر دوم GCC در فصل گذشته، عمل sed زیر برای اجبار ساخت برای استفاده از پرچم کامپایلر -fomit-frame=pointer به‌خاطر حصول اطمینان از استجکام ساخت‌های کامپایلر اعمال می‌شود:

```
case `uname -m` in
i?86) sed -i 's/^T_CFLAGS =$/& -fomit-frame-pointer/' \
gcc/Makefile.in ;;
esac
```

اسکریپت fixincludes به تلاش سهوی گاه و بی‌گاه برای «تعمیر» سرآیندهای سامانه‌ای که تا به حال نصب شده است شهره است. از آن‌جا که می‌دانیم سرآیندهایی که تا به این‌جای کار نصب شده‌اند به تعمیر نیاز ندارند، برای پیش‌گیری از اجرای اجرای این اسکریپت دستور زیر وارد می‌شود:

```
sed -i 's@\./fixinc\.sh@-c true@' gcc/Makefile.in
```

مستندات این بسته ساخت آن را در یک شاخه‌ی ساخت تخصیص داده شده در بیرون شاخه‌ی منبع توصیه می‌کنند:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

آماده سازی برای کامپایل:

```
../gcc-4.6.2/configure --prefix=/usr \
--libexecdir=/usr/lib --enable-shared \
--enable-threads=posix --enable-__cxa_atexit \
--enable-clocale=gnu --enable-languages=c,c++ \
--disable-multilib --disable-bootstrap --with-system-zlib
```

کامپایل بسته:

```
make
```

یک سری از آزمایش‌ها در مجموعه‌ی آزمایشی GCC به خراب کردن پشته شناخته می‌شوند، پس باید پیش از اجرای آزمایش‌ها اندازه‌ی پشته را افزایش داد:

```
ulimit -s 16384
```

بررسی نتیجه:

```
make -k check
```

برای گرفتن خلاصه‌ای از نتایج مجموعه‌ی آزمایشی:

```
../gcc-4.6.2/contrib/test_summary
```

نصب بسته:

```
make install
```

برخی بسته‌ها انتظار دارند پیش‌پردازنده‌ی C در شاخه‌ی lib/ نصب شده باشد. برای پشتیبانی از آن بسته‌ها، این پیوند نرم ساخته می‌شود:

```
ln -sv ../usr/bin/cpp /lib
```

خیلی از بسته‌ها برای فراخوانی کامپایلر C از نام cc استفاده می‌کنند. برای ارضای آن بسته‌ها، یک پیوند نرم ساخته می‌شود:

```
ln -sv gcc /usr/bin/cc
```

حال که زنجیرابزار نهایی در مکان خود قرار گرفته است، مهم است که دوباره مطمئن شویم کامپایل کردن و لینک کردن همان‌گونه که انتظار می‌رود عمل می‌کنند. این کار با انجام همان بررسی‌های سلامتی که پیش‌تر انجام دادیم ممکن است:

```
echo 'main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

حال اطمینان حاصل می‌کنیم که از پرونده‌های شروع صحیحی استفاده می‌کنیم:

```
grep -o '/usr/lib.*/crt[1in].*succeeded' dummy.log
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
/usr/lib/gcc/i686-pc-linux-gnu/4.6.2/../../../crt1.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.6.2/../../../crti.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.6.2/../../../crtn.o succeeded
```

بررسی این که کامپایلر برای پرونده‌های سرآیند درستی جست‌وجو می‌کند:

```
grep -B4 '^ /usr/include' dummy.log
```

این دستور باید با موفقّیت برونده زیر را حاصل دهد:

```
#include <...> search starts here:
/usr/local/include
/usr/lib/gcc/i686-pc-linux-gnu/4.6.2/include
/usr/lib/gcc/i686-pc-linux-gnu/4.6.2/include-fixed
/usr/include
```

سپس باید بررسی شود که لینکر جدید با مسیرهای جست‌وجوی صحیح استفاده می‌شود:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

بعد باید اظمینان پیدا کنیم که داریم از کتاب‌خانه‌ی C درست استفاده می‌کنیم:

```
grep "/lib.*/libc.so.6 " dummy.log
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
attempt to open /lib/libc.so.6 succeeded
```

در پایان باید از استفاده‌ی GCC از لینکر پویای صحیح اطمینان پیدا کرد:

```
grep found dummy.log
```

اگر همه‌چیز درست کار کند نباید خطایی وجود داشته باشد و برونده آخرین دستور به شکل زیر خواهد بود:

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

پس از این که همه‌چیز درست کار کرد، پرونده‌های آزمایشی را تمیز میی کنیم:

```
rm -v dummy.c a.out dummy.log
```

Sed (۱۶-۱-۳

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 8.3 MB

آماده سازی برای کامپایل:

```
./configure --prefix=/usr --bindir=/bin --htmldir=/usr/share/doc/sed-4.2.1
```

کامپایل بسته:

```
make
```

تولید مستندات HTML:

```
make html
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

نصب مستندات HTML:

```
make -C doc install-html
```

۳-۱-۱۷) Bzip2

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 6.4 MB

اعمال وصله‌ای که مستندات را برای این بسته نصب می‌کند:

```
patch -Np1 -i ../bzip2-1.0.6-install_docs-1.patch
```

دستور زیر این اظمینان را می‌دهد که نصب پیوندهای نرم درست است:

```
sed -i 's@\(ln -s -f \)$(PREFIX)/bin/@\1@' Makefile
```

آماده سازی برای کامپایل:

```
make -f Makefile-libbz2_so
make clean
```

کامپایل و بررسی بسته:

```
make
```

نصب بسته:

```
make PREFIX=/usr install
```

نصب دودویی به‌اشتراک گذاشته شده‌ی bzip2 در شاخه‌ی bin/، ایجاد برخی پیوندهای نرم ضروری، و تمیزکاری:

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

۳-۱-۱۸) Ncurses

زمان تقریبی ساخت: 0.8 SBU

فضای دیسک مورد نیاز: 35 MB

آماده سازی برای کامپایل:

```
./configure --prefix=/usr --with-shared --without-debug --enable-widec
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

انتقال کتاب‌خانه‌های به‌اشتراک گذاشته شده به شاخه‌ی lib/ جایی که انتظار می‌رود حضور داشته باشند:

```
mv -v /usr/lib/libncursesw.so.5* /lib
```

خیلی از برنامه‌های کاربردی هنوز از لینکر انتظار دارند که بتواند کتاب‌خانه‌های نویسه‌های غیر عریض Ncurser را پیدا کند. برای پشتیبانی آن‌ها:

```
for lib in ncurses form panel menu ; do \
rm -vf /usr/lib/lib${lib}.so ; \
echo "INPUT(-l${lib}w)" >/usr/lib/lib${lib}.so ; \
ln -sfv lib${lib}w.a /usr/lib/lib${lib}.a ; \
done
ln -sfv libncurses++w.a /usr/lib/libncurses++.a
```

در پایان مطمئن می‌شویم که برنامه‌های قدیمی که هنگام ساخت به دنبال lcurses- می‌گردند هنوط هم قابل ساخت هستند:

```
rm -vf /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" >/usr/lib/libcursesw.so
ln -sfv libncurses.so /usr/lib/libcurses.so
ln -sfv libncursesw.a /usr/lib/libcursesw.a
ln -sfv libncurses.a /usr/lib/libcurses.a
```

در صورت تمایل می‌توان مستندات Ncurses را نصب کرد:

```
mkdir -v
/usr/share/doc/ncurses-5.9
cp -v -R doc/* /usr/share/doc/ncurses-5.9
```

۳-۱-۱۹) Util-linux

زمان تقریبی ساخت: 0.7 SBU

فضای دیسک مورد نیاز: 69 MB

FHS پیشنهاد می‌کند به‌جای شاخه‌ی معمول etc/ از شاخه‌ی var/lib/hwclock/ استفاده شود:

```
sed -e 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
-i $(grep -rl '/etc/adjtime' .)
mkdir -pv /var/lib/hwclock
```

آماده سازی برای کامپایل:

```
./configure --enable-arch --enable-partx --enable-write
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

۳-۱-۲۰) Psmisc

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 3.6 MB

آماده سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

در پایان برنامه‌های killall و fuser را به مکان‌هایی که توسّط FHS مشخّص شده‌اند منتقل می‌کنیم:

```
mv -v /usr/bin/fuser /bin
mv -v /usr/bin/killall /bin
```

E2fsprogs (۲۱-۱-۳)

زمان تقریبی ساخت: 0.5 SBU

فضای دیسک مورد نیاز: 45 MB

مستندات این بسته ساخت آن را در یک شاخه‌ی ساخت تخصیص داده شده در بیرون شاخه‌ی منبع توصیه می‌کنند:

```
mkdir -v build
cd build
```

آماده سازی برای کامپایل:

```
PKG_CONFIG=/tools/bin/true LDFLAGS="-lblkid -luuid" \
../configure --prefix=/usr --with-root-prefix="" \
--enable-elf-shlibs --disable-libblkid --disable-libuuid \
--disable-uuidd --disable-fsck
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب دودویی‌ها، مستندات و کتاب‌خانه‌های به‌اشتراک گذاشته شده:

```
make install
```

نصب کتاب‌خانه‌های ایستا و سرآیندها:

```
make install-libs
```

می‌بایست کتاب‌خانه‌های ایستای نصب‌شده را قابل نوشتن کرد تا علائم رفع‌اشکال بتوانند بعداً برداشته شوند:

```
chmod -v u+w /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

این بسته یک پرونده‌ی info. فشرده‌شده نصب می‌کند ولی پرونده‌ی dir سراسری را به‌روز رسانی نمی‌کند. باید این پرونده را از حالت gzip خارج کرد و سپس با استفاده از دستورات زیر پرونده‌ی dir سامانه را به‌روز رسانی کرد.

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir \
/usr/share/info/libext2fs.info
```

در صورت تمایل می‌توان برخی مستندات اضافی را با اجرای دستورات زیر ایجاد و نصب کرد:

```
makeinfo -o
doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir \
/usr/share/info/com_err.info
```

۳-۱-۲۲) Coreutils

زمان تقریبی ساخت: 3.2 SBU

فضای دیسک مورد نیاز: 99 MB

یک مشکل شناخته‌شده مربوط به برنامه‌ی uname متعلق به این بسته این است که سوییچ p- همواره unknown را برمی‌گرداند. وصله‌ی زیر این رفتار را برای معماری اینتل درست می‌کند:

```
case `uname -m` in
i?86 | x86_64) patch -Np1 -i ../coreutils-8.15-uname-1.patch ;;
esac
```

POSIX نیاز دارد که برنامه‌های Coreutils حتا در منطقه‌های چندبایتی، مرزهای نویسار را تشخیص بدهند. وصله‌ی زیر باگ‌های مربوط به بین‌المللی‌سازی را رفع می‌کند:

```
patch -Np1 -i ../coreutils-8.15-i18n-1.patch
```

آماده سازی برای کامپایل:

```
./configure --prefix=/usr \
--libexecdir=/usr/lib \
--enable-no-install-program=kill,uptime
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

انتقال برنامه‌ها به مکان مشخّص شده توسّط FHS:

```
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
mv -v /usr/share/man/man1/chroot.1 /usr/share/man/man8/chroot.8
sed -i s/\"1\"/\"8\"/1 /usr/share/man/man8/chroot.8
```

برخی اسکریپت‌ها در بسته‌ی LFS-Bootscripts وابسته به head، sleep و nice هستند. از آن‌جا که ممکن است در طول مراحل نخستین راه‌اندازی، usr/ در دست‌رس نباشد، آن دودویی‌ها باید روی پارتیشن ریشه قرار بگیرند:

```
mv -v /usr/bin/{head,sleep,nice} /bin
```

۳-۱-۲۳) Iana-Etc

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 2.3 MB

دستور زیر داده‌ی خامی که توسّط IANA فراهم شده را به قالب‌های درست برای پرونده‌های داده‌ای etc/protocols/ و etc/services/ تبدیل می‌کند:

```
make
```

این بسته مجموعه‌ی آزمایشی ندارد و این‌گونه نصب می‌شود:

```
make install
```

M4 (۲۴-۱-۳)

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 14.2 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

برای بررسی نتیجه، نخست یک برنامه‌ی آزمایشی را تصحیح می‌کنیم و سپس برنامه‌های آزمایشی را اجرا می‌کنیم:

```
sed -i -e '41s/ENOENT/& || errno == EINVAL/' tests/test-readlink.h
make check
```

نصب بسته:

```
make install
```

Bison (۲۵-۱-۳)

زمان تقریبی ساخت: 1.1 SBU

فضای دیسک مورد نیاز: 19.2 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

سامانه‌ی پیکربندی موجب می‌شود اگر یک برنامه‌ی bison از پیش در $PATH نباشد، Bison بدون پشیبانی از بین‌المللی‌سازی پیام‌های خطا ساخته شود. افزونه‌ی زیر این مشکل را درست می‌کند:

```
echo '#define YYENABLE_NLS 1' >> lib/config.h
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 0.5 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۲۶) Procps

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 2.3 MB

اعمال وصله‌ای برای پیش‌گیری از نمایش یک پیام خطا هنگام تعیین سرعت ساعت هسته:

```
patch -Np1 -i ../procps-3.2.8-fix_HZ_errors-1.patch
```

اعمال وصله‌ای برای تعمیر یک مشکل مربوط به یونی‌کد در برنامه‌ی watch:

```
patch -Np1 -i ../procps-3.2.8-watch_unicode-1.patch
```

رفع مشکلی در Makefile که مانع از ساخت procps توسط make-3.82 می‌شود:

```
sed -i -e 's@\*/module.mk@proc/module.mk ps/module.mk@' Makefile
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

۳-۱-۲۷) Grep

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 23 MB

ابتدا باید مشکل کوچیکی با یک اسکریپت آزمایشی را رفع کرد:

```
sed -i 's/cp/#&/' tests/unibyte-bracket-expr
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --bindir=/bin
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

Readline (۲۸-۱-۳)

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 13.8 MB

نصب دوباره‌ی Readline موجب می‌شود کتاب‌خانه‌های قدیمی به libraryname>.old< منتقل شوند. با این که چنین چیزی در خالت عادی مشکل نیست، در برخی موارد می‌تواند موجب یک باگ لینک کردن در ldconfig شود. با دو دستور sed زیر می‌توان از این عمل جلوگیری کرد:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

اعمال وصله‌ای که یک باگ ناشناخته از بالادست تعمیر شده است را درست می‌کند:

```
patch -Np1 -i ../readline-6.2-fixes-1.patch
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --libdir=/lib
```

کامپایل بسته:

```
make SHLIB_LIBS=-lncurses
```

نصب بسته:

```
make install
```

انتقال کتاب‌خانه‌ها به یک مکان مرتبط‌تر:

```
mv -v /lib/lib{readline,history}.a /usr/lib
```

برداشتن پرونده‌های so. از lib/ و بازپیوند آن‌ها در usr/lib/

```
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.6 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.6 /usr/lib/libhistory.so
```

در صورت تمایل می‌توان مستندات را نیز نصب کرد:

```
mkdir -v /usr/share/doc/readline-6.2
install -v -m644 doc/*.{ps,pdf,html,dvi} \
/usr/share/doc/readline-6.2
```

۳-۱-۲۹) Bash

زمان تقریبی ساخت: 1.4 SBU

فضای دیسک مورد نیاز: 35 MB

اعمال وصله‌ای که باگ‌های زیادی که از بالادست گزارش شده‌اند را تعمیر می‌کند:

```
patch -Np1 -i ../bash-4.2-fixes-4.patch
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --bindir=/bin \
--htmldir=/usr/share/doc/bash-4.2 --without-bash-malloc \
--with-installed-readline
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

جایگزینی با برنامه‌ی bash تازه کامپایل شده:

```
exec /bin/bash --login +h
```

۳-۱-۳۰) Libtool

زمان تقریبی ساخت: 3.7 SBU

فضای دیسک مورد نیاز: 35 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 3 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۳۱) GDBM

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 2.7 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --enable-libgdbm-compat
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 3 SBU):

```
make check
```

نصب بسته:

```
make install
```

Inetutils (۳۲-۱-۳)

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 17 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
--localstatedir=/var --disable-ifconfig \
--disable-logger --disable-syslogd --disable-whois \
--disable-servers
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 3 SBU):

```
make check
```

نصب بسته:

```
make install
make -C doc html
make -C doc install-html docdir=/usr/share/doc/inetutils-1.9.1
```

انتقال برخی برنامه‌ها به مکان سازکار با FHS:

```
mv -v /usr/bin/{hostname,ping,ping6} /bin
mv -v /usr/bin/traceroute /sbin
```

Perl (۳۳-۱-۳)

زمان تقریبی ساخت: 7.6 SBU

فضای دیسک مورد نیاز: 235 MB

ابتدا باید یک پرونده‌ی پایه‌ای /etc/hosts ساخت تا در یکی از پرونده‌های پیکربندی پرل به آن مراجعه شود:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

اعمال وصله‌ای که یک آسیب‌پذیری امنیتی در پرل را درست می‌کند:

```
patch -Np1 -i ../perl-5.14.2-security_fix-1.patch
```

به صورت پیش‌فرض پرل از یک رونوشت داخلی Zlib برای ساخت استفاده میکند. با دستور زیر می‌توان به پرل گفت که از کتاب‌خانه‌ی Zlib نصب شده استفاده کند:

```
sed -i -e "s|BUILD_ZLIB\s*= True|BUILD_ZLIB = False|" \
-e "s|INCLUDE\s*= ./zlib-src|INCLUDE = /usr/include|" \
-e "s|LIB\s*= ./zlib-src|LIB = /usr/lib|" \
cpan/Compress-Raw-Zlib/config.in
```

آماده‌سازی برای کامپایل:

```
sh Configure -des -Dprefix=/usr \
-Dvendorprefix=/usr \
-Dman1dir=/usr/share/man/man1 \
-Dman3dir=/usr/share/man/man3 \
-Dpager="/usr/bin/less -isR" \
-Duseshrplib
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 2.5 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۳۴) Autoconf

زمان تقریبی ساخت: 4.8 SBU

فضای دیسک مورد نیاز: 12.4 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 4.7 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۳۵) Automake

زمان تقریبی ساخت: 18.3 SBU

فضای دیسک مورد نیاز: 28.8 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.11.3
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 10 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۳۶) Difutils

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 6.3 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۳۷) Gawk

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 28 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --libexecdir=/usr/lib
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

در صورت تمایل می‌توان مستندات را نصب کرد:

```
mkdir -v /usr/share/doc/gawk-4.0.0
cp
-v doc/{awkforai.txt,*.{eps,pdf,jpg}} \
/usr/share/doc/gawk-4.0.0
```

۳-۱-۳۸) findutils

زمان تقریبی ساخت: 0.5 SBU

فضای دیسک مورد نیاز: 22 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --libexecdir=/usr/lib/findutils \
--localstatedir=/var/lib/locate
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

برخی اسکریپت‌ها در بسته‌ی اسکریپت‌های راه‌اندازی به find وابسته‌اند. از آن‌جا که ممکن است در طول مراحل نخستین راه‌اندازی usr/ در دست‌رس نباشد، این برنامه نیاز دارد روی پارتیشن ریشه قرار بگیرد. هم‌چنین اسکریپت updatedb باید دستکاری شود تا یک مسیر صریح را درست کند:

```
mv -v /usr/bin/find /bin
sed -i 's/find:=${BINDIR}/find:=\/bin/' /usr/bin/updatedb
```

Flex (۳۹-۱-۳

زمان تقریبی ساخت: 0.7 SBU

فضای دیسک مورد نیاز: 28 MB

اعمال وصله‌ای که یک باگ در مولّد پویشگر C++ را رفع می‌کند که موجب می‌شود هنگام استفاده از GCC-4.6.2، کامپایل پویشگر شکست بخورد:

```
patch -Np1 -i ../flex-2.5.35-gcc44-1.patch
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

هنوز چندتا از برنامه‌ها راجع‌به flex چیزی نمی‌دانند و تلاش می‌کنند سلف آن، lex را اجرا کنند. برای پشتیبانی از این برنامه‌ها اسکریپتی به نام lex می‌نویسیم که flex را در حالت شبیه‌سازی lex فراخوانی کند:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex
exec /usr/bin/flex -l "$@"
# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

در صروت تمایل می‌توان پرونده‌ی مستندات flex.pdf را نصب نمود:

```
mkdir -v /usr/share/doc/flex-2.5.35
cp -v doc/flex.pdf \
/usr/share/doc/flex-2.5.35
```

۳-۱-۴۰) Gettext

زمان تقریبی ساخت: 5.8 SBU

فضای دیسک مورد نیاز: 125 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr \
--docdir=/usr/share/doc/gettext-0.18.1.1
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 3 SBU):

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۴۱) Groff

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 78 MB

آماده‌سازی برای کامپایل:

```
PAGE=A4 ./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

برخی برنامه‌های مستندات مانند xman بدون پیوندهای نرم زیر درست کار نخواهند کرد:

```
ln -sv eqn /usr/bin/geqn
ln -sv tbl /usr/bin/gtbl
```

۳-۱-۴۲) Xz

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 13 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --libdir=/lib --docdir=/usr/share/doc/xz-5.0.3
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make pkgconfigdir=/usr/lib/pkgconfig install
```

GRUB (۴۳-۱-۳

زمان تقریبی ساخت: 0.6 SBU

فضای دیسک مورد نیاز: 76 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr \
--sysconfdir=/etc \
--disable-grub-emu-usb \
--disable-efiemu \
--disable-werror
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make pkgconfigdir=/usr/lib/pkgconfig install
```

Gzip (۴۴-۱-۳

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 3.3 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --bindir=/bin
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

انتقال برخی برنامه‌ها که نیازی نیست در فایل‌سیستم ریشه باشند:

```
mv -v /bin/{gzexe,uncompress,zcmp,zdiff,zegrep} /usr/bin
mv -v /bin/{zfgrep,zforce,zgrep,zless,zmore,znew} /usr/bin
```

IPRoute2 (۴۵-۱-۳

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 6.6 MB

دودویی arpd که در این بسته آمده است وابسته به پایگاه داده‌ی برکلی است. از آن‌جا که یک سامانه‌ی لینوکسی پایه نیاز چندانی به arpd ندارد، با دستور زیر می‌توان این وابستگی به پایگاه داده‌ی برکلی را از میان برد:

```
sed -i '/^TARGETS/s@arpd@@g' misc/Makefile
sed -i /ARPD/d Makefile
rm man/man8/arpd.8
```

برداشتن مراجع به برخی سرآیندهای Libnl که IPRoute2 نیازی به آن‌ها ندارد:

```
sed -i -e '/netlink\//d' ip/ipl2tp.c
```

کامپایل بسته:

```
make DESTDIR=
```

نصب بسته:

```
make DESTDIR= MANDIR=/usr/share/man \
DOCDIR=/usr/share/doc/iproute2-3.2.0 install
```

Kbd (۴۶-۱-۳

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 16.0 MB

رفتار دکمه‌های پس‌بر و حذف با نقشه‌های کلید موجود در این بسته سازگار نیست. وصله‌ی زیر را برای این مورد اعمال می‌کنیم:

```
patch -Np1 -i ../kbd-1.15.2-backspace-1.patch
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --datadir=/lib/kbd
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

برخی اسکریپت‌ها در بسته‌ی اسکریپت‌های راه‌اندازی وابسته به kbd_mode، loadkeys، openvt و setfont هستند. از آن‌جا که ممکن است در طول مراحل نخستین راه‌اندازی /usr در دست‌رس نباشد، این دودویی‌ها روی پارتیشن ریشه قرار بگیرند:

```
mv -v /usr/bin/{kbd_mode,loadkeys,openvt,setfont} /bin
```

در صورت تمایل می‌توان مستندات را نیز نصب کرد:

```
mkdir -v /usr/share/doc/kbd-1.15.2
cp -R -v doc/* \
/usr/share/doc/kbd-1.15.2
```

۳-۱-۴۷) Kmod

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 16.0 MB

آماده‌سازی برای کامپایل:

```
liblzma_CFLAGS="-I/usr/include" \
liblzma_LIBS="-L/lib -llzma" \
zlib_CFLAGS="-I/usr/include" \
zlib_LIBS="-L/lib -lz" \
./configure --prefix=/usr --bindir=/bin --libdir=/lib --sysconfdir=/etc \
--with-xz --with-zlib
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته و ایجاد پیوندهای نرم برای سازگاری با Module-Init-Tools، بسته‌ای که پیش‌تر ماژول‌های هسته‌ی لینوکس را اداره می‌کرد:

```
make pkgconfigdir=/usr/lib/pkgconfig install
for target in depmod insmod modinfo modprobe rmmod; do
ln -sv ../bin/kmod /sbin/$target
done
ln -sv kmod /bin/lsmod
```

۳-۱-۴۸) Less

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 3.5 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --sysconfdir=/etc
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

۳-۱-۴۹) Libpipeline

زمان تقریبی ساخت: 0.1 SBU

فضای دیسک مورد نیاز: 8.0 MB

آماده‌سازی برای کامپایل:

```
./configure CHECK_CFLAGS=-I/tools/include \
CHECK_LIBS="-L/tools/lib -lcheck" --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۵۰) Make

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 9.7 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

۳-۱-۵۱) Man

زمان تقریبی ساخت: 0.4 SBU

فضای دیسک مورد نیاز: 22 MB

آماده‌سازی برای کامپایل:

```
PKG_CONFIG=/tools/bin/true \
libpipeline_CFLAGS='' \
libpipeline_LIBS='-lpipeline' \
./configure --prefix=/usr --libexecdir=/usr/lib \
--docdir=/usr/share/doc/man-db-2.6.1 --sysconfdir=/etc \
--disable-setuid --with-browser=/usr/bin/lynx \
--with-vgrind=/usr/bin/vgrind --with-grap=/usr/bin/grap
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

Patch (۵۲-۱-۳

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 1.9 MB

اعمال وصله‌ای که از اجرای یکی از آزمایش‌های مجموعه‌ی آزمایشی که به ed نیاز دارد جلوگیری میکند:

```
patch -Np1 -i ../patch-2.6.1-test_fix-1.patch
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

Shadow (۵۳-۱-۳

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 30 MB

اعمال وصله‌ای که مشکلی را هنگام اجرای برنامه‌های مختلفی از این بسته وقتی دیمون nscd در حال اجرا نباشد رخ می‌دهد، درست می‌کند:

```
patch -Np1 -i ../shadow-4.1.5-nscd-1.patch
```

از آن‌جا که Coreutils نسخه‌ی بهتری از برنامه‌ی group و صفحات راهنمای آن فراهم می‌کند، نصب آن را غیرفعّال می‌کنیم:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile.in
find man -name Makefile.in -exec sed -i 's/groups\.1 / /' {} \;
```

به جای استفاده از رمزگذاری پیش‌فرض برای گذرواژه‌ها از روش امن‌تر SHA-512 استفاده می‌کنیم:

```
sed -i -e 's@#ENCRYPT_METHOD DES@ENCRYPT_METHOD SHA512@' \
-e 's@/var/spool/mail@/var/mail@' etc/login.defs
```

آماده‌سازی برای کامپایل:

```
./configure --sysconfdir=/etc
```

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

انتقال یک برنامه‌ی جا انداخته‌شده به مکان صحیحش:

```
mv -v /usr/bin/passwd /bin
```

برای فعّال‌سازی گذرواژهای shaddow شده:

```
pwconv
```

برای فعّال سازی گذرواژه‌های گروهی shaddow شده:

```
grpconv
```

تنظیم گذرواژه‌ی ریشه:

```
passwd root
```

۳-۱-۵۴) Sysklogd

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 0.5 MB

کامپایل بسته:

```
make
```

نصب بسته:

```
make install
```

ایجاد یک پرونده‌ی پیکربندی جدید:

```
cat > /etc/syslog.conf << "EOF"
# Begin /etc/syslog.conf
auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *
# End /etc/syslog.conf
EOF
```

۳-۱-۵۵) Sysvinit

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 1 MB

وقتی سطوح اجرا تغییر می‌کنند (مثلا هنگام خاموش کردن سامانه)، init سیگنال خاتمه را به پردازه‌هایی که خودش شروع کرده و نباید در سطح اجرای جدید در حال اجرا باشند ارسال می‌کند. در حین این کار، init پیام‌هایی برون می‌دهد که گویی این سیگنال‌ها را برای همه‌ی پردازه‌های در حال اجرا فرستاده. برای پیش‌گیری از چنین سوء تعبیری منبع را چنان تغییر می‌دهیم که این پیام‌ها به درستی نشان داده شوند:

```
sed -i 's@Sending processes@& configured via /etc/inittab@g' \
src/init.c
```

نسخه‌های جدید تری از برنامه‌های wall و mountpoint پیش‌تر توسّط Util-linux نصب شدند. با این دستور از نصب نسخه‌ی sysvinit آن‌ها و صفحات راهنمایشان جلوگیری می‌کنیم:

```
sed -i -e 's/utmpdump wall/utmpdump/' \
-e '/= mountpoint/d' \
-e 's/mountpoint.1 wall.1//' src/Makefile
```

کامپایل بسته:

```
make -C src
```

نصب بسته:

```
make -C src install
```

۳-۱-۵۶) Tar

زمان تقریبی ساخت: 1.9 SBU

فضای دیسک مورد نیاز: 21.2 MB

آماده‌سازی برای کامپایل:

```
FORCE_UNSAFE_CONFIGURE=1 ./configure --prefix=/usr \
--bindir=/bin --libexecdir=/usr/sbin
```

کامپایل بسته:

```
make
```

بررسی نتیجه (حدود 1 SBU):

```
make check
```

نصب بسته:

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.26
```

۳-۱-۵۷) Texinfo

زمان تقریبی ساخت: 0.3 SBU

فضای دیسک مورد نیاز: 21 MB

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

به دل‌خواه می‌توان کامپوننت‌های متعلق به نصب TEXرا نصب کرد:

```
make TEXMF=/usr/share/texmf install-tex
```

Udev (۵۸-۱-۳

زمان تقریبی ساخت: 0.2 SBU

فضای دیسک مورد نیاز: 9.3 MB

تربال udev-config شامل پرونده‌هایی است که برای پیکر بندی Udev به کار می‌روند. آن را درون شاخه‌ی منبع می‌گشاییم:

```
tar -xvf ../udev-config-20100128.tar.bz2
```

چند دستگاه و شاخه که Udev به خاطر این که آن‌ها خیلی زود در فرایند راه‌اندازی مورد نیاز هستند، نمی‌تواند اداره‌شان کند را ایجاد می‌کنیم:

```
install -dv /lib/{firmware,udev/devices/pts}
mknod -m0666 /lib/udev/devices/null c 1 3
```

آماده‌سازی برای کامپایل:

```
BLKID_CFLAGS="-I/usr/include/blkid" \
BLKID_LIBS="-L/lib -lblkid" \
KMOD_CFLAGS="-I/usr/include" \
KMOD_LIBS="-L/lib -lkmod" \
./configure --prefix=/usr \
--with-rootprefix='' \
--bindir=/sbin \
--sysconfdir=/etc \
--libexecdir=/lib \
--enable-rule_generator \
--disable-introspection \
--disable-keymap \
--disable-gudev \
--with-usb-ids-path=no \
--with-pci-ids-path=no \
--with-systemdsystemunitdir=no
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make check
```

نصب بسته:

```
make install
```

برداشتن یک شاخه‌ی مستندات خالی:

```
rmdir -v /usr/share/doc/udev
```

نصب پرونده‌های قواعد:

```
cd udev-config-20100128
make install
```

نصب مستندات پرونده‌های قواعد:

```
make install-doc
```

۳-۱-۵۹) Vim

زمان تقریبی ساخت: 1.0 SBU

فضای دیسک مورد نیاز: 87 MB

نخست مکان پیش‌فرض پرونده‌ی پیکر بندی vimrc را به etc/ تغییر می‌دهیم:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

آماده‌سازی برای کامپایل:

```
./configure --prefix=/usr --enable-multibyte
```

کامپایل بسته:

```
make
```

بررسی نتیجه:

```
make test
```

نصب بسته:

```
make install
```

خیلی از کاربران عادت به استفاده از vi به جای vim دارند. برای اجرای vim وقتی کاربران از روی عادت vi را وارد می‌کنند، پیوند نرمی برای دودویی و صفحه‌ی راهنما در زبان‌های فراهم شده ایجاد می‌کنیم:

```
ln -sv vim /usr/bin/vi
for L in /usr/share/man/{,*/}man1/vim.1; do
ln -sv vim.1 $(dirname $L)/vi.1
done
```

به صورت پیش‌فرض مستندات Vim در /usr/share/vim/ نصب می‌شوند. پیوند نرم زیر اجازه می‌دهد مستندات از /usr/share/doc/vim-7.3 دست‌رس پذیر باشند تا با سایر برنامه‌ها هماهنگ شود:

```
ln -sv ../vim/vim73/doc /usr/share/doc/vim-7.3
```

برای ایجاد پرونده‌ی پیکربندی:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc
set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
set background=dark
endif
" End /etc/vimrc
EOF
```

مستندات برای دیگر انتخاب‌های در دست‌رس می‌توانند با اجرای دستور زیر به‌دست آیند:

```
vim -c ':options'
```

۳-۱-۶۰) تُنُک‌سازی دوباره

اگر سیستم‌عامل نهایی جهت مصارق اشکال‌زدایی از نرم‌افزارهای سیستمی به کار نمی‌رود، می‌توان با برداشتن علائم اشکال‌زدایی از دودویی‌ها و کتاب‌خانه‌ها اندازه‌ی سامانه را حدود ۹۰ مگابایت

کاهش داد. این کار هیچ ضرری ندارد، به جز این که دیگر قادر به اشکال زدایی کامل از برنامه‌هایی که هم‌اکنون روی سامانه نصب شده‌اند نخواهیم بود.

پیش از اجرای این تنک‌سازی باید توجّه ویژه‌ای نمود که هیچ‌یک از دودویی‌هایی که می‌خواهند تنک شوند درحال اجرا نیستند. برای حصول اطمینان می‌توان ابتدا از محیط chroot خارج شد:

```
logout
```

سپس دوباره با این دستور وارد شد:

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

حال دودویی‌ها و کتاب‌خانه‌ها می‌توانند بدون خطر تنک شوند:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
-exec /tools/bin/strip --strip-debug '{}' ';'
```

۳-۱-۶۱) پاک‌سازی

از این به بعد پس از خروج با دستور زیر وارد محیط chroot می‌شویم:

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

دلیل این کار این است که دیگر به برنامه‌های موقّتی موجود در /tools نیازی نیست و می‌توان در صورت تمایل آن‌ها را پاک نمود.

۳-۲) تنظیم کردن اسکریپت‌های راه‌اندازی

۳-۲-۱) پیکربندی عمومی شبکه

ایجاد نام‌های پایدار برای واسط‌های شبکه

کارت‌های شبکه در سامانه به‌ترتیب با شماره‌ای خاص شناخته می‌شوند. در برخی موارد ممکن است پس از راه‌اندازی مجدّد جای این شماره‌ها با هم عوض شود که موجب بروز مشکل در کار دستگاه‌ها می‌شود. برای جلوگیری از چنین امری از دستور زیر استفاده می‌کنیم که به سامانه می‌گوید شماره‌ی دستگاه‌ها را برای هر آدرس سخت‌افزاری به صورت یکتا درنظر بگیرد:

```
for NIC in /sys/class/net/* ; do
INTERFACE=${NIC##*/} udevadm test --action=add $NIC
done
```

ایجاد پرونده‌های پیکربندی واسط‌های شبکه

هر واسطی در شبکه با یک پرونده در مسیر /etc/sysconfig پیکربندی می‌شود. برای مثال در این مورد خاص، واسط شبکه‌ی eth1 به صورت زیر پیکربندی شد:

```
cd /etc/sysconfig/
cat > ifconfig.eth1 << "EOF"
ONBOOT=yes
IFACE=eth1
SERVICE=ipv4-static
IP=192.168.56.101
GATEWAY=192.168.56.1
PREFIX=24
BROADCAST=192.168.56.255
EOF
```

ایجاد پرونده‌ی /etc/resolve.conf

این پرونده برای تعیین کردن آدرس سرویس‌دهنده‌ی DNS به کار می‌رود. برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf
domain <Your Domain Name>
nameserver 4.2.2.4
```

```
nameserver 8.8.8.8
# End /etc/resolv.conf
EOF
```

۳-۲-۲) شخصی‌سازی پرونده‌ی /etc/hosts

این پرونده به سامانه می‌گوید که برای دسترسی به هر میزبان باید به کدام آدرس برود. یکی از فواید پیکربندی این پرونده این است که به ما اجازه می‌دهد به جای هربار وارد کردن آدرس سیستم جاری از نام آن استفاده کنیم. برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts
127.0.0.1 localhost
# End /etc/hosts
EOF
```

۳-۲-۳) LFS-Bootscripts

زمان تقریبی ساخت: کم‌تر از 0.1 SBU

فضای دیسک مورد نیاز: 260 KB

نصب بسته:

```
make install
```

۳-۲-۴) پیکربندی sysvinit

نخستین برنامه‌ای که در زمان راه‌اندازی هسته اجرا می‌شود init است. این برنامه پرونده‌ی /etc/inittab را می‌خواند. این پرونده را می‌توان به صورت زیر ایجاد کرد:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab
id:3:initdefault:
si::sysinit:/etc/rc.d/init.d/rc S
l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
```

```
l6:6:wait:/etc/rc.d/init.d/rc 6
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
su:S016:once:/sbin/sulogin
1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600
# End /etc/inittab
EOF
```

این پرونده به rc دستور می‌دهد که تمام اسکریپت‌هایی که در /etc/rc.d/rcsysinit.d با حرف S شروع می‌شوند را اجرا کند.

۳-۲-۵) پیکربندی نام میزبان سامانه

بخشی از وظیفه‌ی اسکریپت localnet تنظیم نام میزبان سامانه است. این نام باید در پرونده‌ی /etc/sysconfig/network پیکربندی شود.

برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
echo "HOSTNAME=Danial-Project" > /etc/sysconfig/network
```

۳-۲-۶) پیکربندی اسکریپت setclock

اسکریپت setclock زمان را از ساعت سخت‌افزاری می‌خواند. اگر این ساعت بر روی UTC تنظیم شده باشدبا استفاده از پرونده‌ی /etc/localtime آن را به زمان محلّی تبدیل می‌کند. از آن‌جا که راهی برای فهمیدن خودکار این که ساعت سخت‌افزاری بر روی UTC تنظیم شده یا خیر وجود ندارد، این مطلب را با ساخت پرونده‌ی زیر به اطلاع این اسکریپت می‌رسانیم:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock
UTC=1
# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=
# End /etc/sysconfig/clock
EOF
```

۳-۲-۷) پیکربندی پیشانه‌ی لینوکس

اسکریپت console پرونده‌ی /etc/sysconfig/console را برای اطلاعات پیکربندی خوانده و تصمیم می‌گیرد که کدام نقشه‌ی کلید و قلم صفحه باید انتخاب شود.

برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"
# End /etc/sysconfig/console
EOF
```

۳-۲-۸) پرونده‌ی rc.site

این پرونده می‌تواند به عنوان جایگزین تنظیمات همه‌ی اسکریپت‌های راه‌اندازی به کار رود. اگر متغیّرهایی در هر دو جا تعریف شده باشند، مقادیر مشخّص شده در پرونده‌های خاص مربوط به اسکریپت‌ها ارجح است. هم‌چنین این پرونده شامل پارامترهایی است که می‌توانند دیگر جنبه‌های فرایند راه‌اندازی را شخصی‌سازی کنند.

پیکربندی این پرونده در پیوست ب آمده است.

۳-۲-۹) پرونده‌های شروع پوسته‌ی bash

وقتی bash به عنوان یک پوسته‌ی فعال احضار می‌شود، پرونده‌های /etc/profile و .bash_profile/ خوانده می‌شوند. این پرونده در پایه‌ای‌ترین حالت متغیّرهای مربوط به منطقه‌ها را تنظیم می‌کنند.

برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile
export LANG=fa_IR.utf8
# End /etc/profile
EOF
```

۳-۲-۱۰) ایجاد پرونده‌ی /etc/inputrc

پرونده‌ی inputrc مسیردهی صفحه‌کلید را در شرایط خاص برعهده می‌گیرد. این پرونده، پیپرونده‌ی شروعی است که توسّط Readline – کتاب‌خانه‌ای مربوط به ورودی که توسط bash و بیش‌تر پوسته‌های دیگر استفاده می‌شود – به کار می‌رود.

با دستور زیر این پرونده را به صورت آن‌چه که عموماً مرسوم است ایجاد می‌کنیم:

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>
# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off
# Enable 8bit input
set meta-flag On
set input-meta On
# Turns off 8th bit stripping
set convert-meta Off
# Keep the 8th bit for display
set output-meta On
# none, visible or audible
set bell-style none
# All of the following map the escape sequence of the value
# contained in the 1st argument to the readline specific functions
"\eOd": backward-word
"\eOc": forward-word
# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert
# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line
# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line
# End /etc/inputrc
EOF
```

۳-۳) قابل راه‌اندازی کردن سیستم‌عامل جدید

۳-۳-۱) ایجاد پرونده‌ی /etc/fstab

پرونده‌ی fstab توسّط برخی برنامه‌ها برای تعیین این که به صورت پیش‌فرض فایل‌سیستم‌ها باید کجا سوار شوند، به چه ترتیبی سوار شوند و کدام‌یک باید پیش از سوار شدن بررسی شوند، به کار برده می شود.

برای مثال در این مورد خاص، این پرونده به صورت زیر پیکربندی شد:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab
# file system mount-point type        options               dump fsck-order
/dev/sda3        /        ext4        defaults               1        1
/dev/sda4        swap     swap        pri=1                  0        0
proc             /proc    proc        nosuid,noexec,nodev    0        0
sysfs            /sys     sysfs       nosuid,noexec,nodev    0        0
devpts           /dev/pts devpts      gid=4,mode=620         0        0
tmpfs            /run     tmpfs       defaults               0        0
devtmpfs         /dev     devtmpfs    mode=0755,nosuid       0        0
# End /etc/fstab
EOF
```

۳-۳-۲) Linux

زمان تقریبی ساخت: 1.0 - 5.0 SBU

فضای دیسک مورد نیاز: 540 - 800 MB

نصب هسته

نصب هسته شامل چند مرحله می‌شود: پیکربندی، کامپایل و نصب.

آماده‌سازی برای کامپایل:

```
make mrproper
```

پیکربندی هسته:

```
make menuconfig
```

کامپایل بسته:

```
make
```

نصب ماژول‌های هسته:

```
make modules_install
```

رونوشت از تصویر هسته به شاخه‌ی boot:

```
cp -v arch/x86/boot/bzImage /boot/vmlinuz-3.2.6-danial
```

نگه‌داری پرونده‌ی علائم هسته:

```
cp -v System.map /boot/System.map-3.2.6
```

نگه‌داری پرونده‌ی پیکربندی هسته:

```
cp -v .config /boot/config-3.2.6
```

نصب مستندات هسته:

```
install -d /usr/share/doc/linux-3.2.6
cp -r Documentation/* /usr/share/doc/linux-3.2.6
```

پیکربندی ترتیب بار شدن ماژول‌های هسته

از آن‌جا که راه‌اندازهای USB باید به ترتیب خاصی بار شوند، به وسیله‌ی پرونده‌ی زیر ترتیب آن‌ها را مشخّص می‌کنیم:

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# Begin /etc/modprobe.d/usb.conf
install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true
# End /etc/modprobe.d/usb.conf
EOF
```

۳-۳-۳) استفاده از GRUB برای تنظیم فرایند راه‌اندازی

تنظیم کردن پیکربندی

گراب با نوشتن داده‌ها روی نخستین شیار فیزیکی دیسک سخت کار می‌کند. این ناحیه جزو هیچ فایل‌سیستمی نیست. برنامه‌های آن‌جا به ماژول‌های گراب در پارتیشن boot دست‌رسی پیدا می‌کنند که به صورت پیش‌فرض در /boot/grub قرار دارند.

نصب پرونده‌های گراب در boot/grub/ و تنظیم شیار راه‌اندازی:

```
grub-install /dev/sda
```

ایجاد پرونده‌ی پیکربندی

با دستور زیر پرونده‌ی پیکربندی را مطابق با شرایط خود ایجاد می‌کنیم:

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5
insmod ext2
set root=(hd0,2)
menuentry "Danial GNU/Linux" {
linux
/boot/vmlinuz-3.2.6-danial root=/dev/sda3 ro
}
EOF
```

۳-۴) پایان

۳-۴-۱) راه‌اندازی مجدّد سامانه

حال که سیستم‌عامل جدیدبه درستی نصب شد می‌توانیم از محیط chroot بیرون آمده و با راه‌اندازی مجدد سامانه، به درون سیستم‌عامل شخصی خودمان برویم. برای بیرون آمدن از محیط chroot:

```
logout
```

سپس باید فایل‌سیستم‌هایی را که سوار کرده بودیم آزاد کنیم:

```
umount -v $LFS/dev/pts
umount -v $LFS/dev/shm
umount -v $LFS/dev
umount -v $LFS/proc
umount -v $LFS/sys
```

و در نهایت خود فایل‌سیستم LFS را آزاد می‌کنیم:

```
umount -v $LFS
```

حال با این دستور سامانه را راه‌اندازی مجدّد می‌کنیم:

```
shutdown -r now
```

فصل چهارم

بحث و نتیجه‌گیری

۴-۱) نتیجه‌گیری

دیدیم که چه‌گونه می‌توان با کمی تلاش و دانش، سیستم‌عاملی ایجاد کرد که تک‌تک بسته‌های ان را خودمان بنا به نیازهای خود پیکربندی کرده باشیم و همه‌ی آن‌ها جزو نرم‌افزارهای آزاد باشند.

استفاده از چنین سیستم‌عامل‌هایی نه‌تنها به شناخت کامل ما از رایانه به عنوان ابزاری برای برآورده ساختن نیازهای محاسباتی ما کمک شایانی می‌کند، بلکه هم در امر سرعت و هم در بحث امنیت اطلاّعات به نوعی تنها راه چاره‌ی ما نیز می‌باشد.

۴-۲) پیشنهادات

می‌توان برای سریع‌تر شدن فرایند راه‌اندازی سیستم‌عامل از یک فایل سیستم مجازی استفاده کرد و آن را با عبارت «initrd» به GRUB معرّفی کرد.

هم‌چنین می‌توان برای ایجاد تغییرات دل‌خواه در bash برای شخصی سازی محیط آن، پیکربندی‌ها را در یک پرونده ذخیره کرد و آن را با عبارت «source» به پرونده‌ی نمایه‌ی پوسته در آدرس /etc/profile معرّفی کرد.

پیوست‌ها

پیوست الف) لیست وصله‌های مورد نیاز

• Bash Upstream Fixes Patch - 22 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/bash-4.2-fixes-4.patch

MD5 sum: 244e3ff74d53792f1db32dea75dc8627

• Bzip2 Documentation Patch - 1.6 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/bzip2-1.0.6-install_docs-1.patch

MD5 sum: 6a5ac7e89b791aae556de0f745916f7f

• Coreutils Internationalization Fixes Patch - 123 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/coreutils-8.15-i18n-1.patch

MD5 sum: 70953451fa1d0e950266b3d0477adb8d

• Coreutils Uname Patch - 1.6 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/coreutils-8.15-uname-1.patch

MD5 sum: 500481b75892e5c07e19e9953a690e54

• Flex GCC-4.4.x Patch - 1 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/flex-2.5.35-gcc44-1.patch

MD5 sum: ad9109820534278c6dd0898178c0788f

• GCC Cross Compile Patch - 1.8 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/gcc-4.6.2-cross_compile-1.patch

MD5 sum: 1b7886a7a4df3a48617e88a481862264

• GCC Startfiles Fix Patch - 1.5 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/gcc-4.6.2-startfiles_fix-1.patch

MD5 sum: 799ef1971350d2e3c794f2123f247cc6

• Glibc Bug Fixes Patch - 5.5 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/glibc-2.14.1-fixes-1.patch

MD5 sum: 13bdfb7db1654d9c3d7934d24479a6c4

• Glibc Bug Sort Relocatable Objects Patch - 8.0 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/glibc-2.14.1-sort-1.patch

MD5 sum: 740e71017059a4290761db0cc9dd63f3

• Glibc GCC Build Fix Patch - 2.5 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/glibc-2.14.1-gcc_fix-1.patch

MD5 sum: d1f28cb98acb9417fe52596908bbb9fd

• Glibc GCC CPUID Patch - 0.8 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/glibc-2.14.1-cpuid-1.patch

MD5 sum: 4f110dc9c8d4754fbda841492ce796b4

• Kbd Backspace/Delete Fix Patch - 12 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/kbd-1.15.2-backspace-1.patch

MD5 sum: f75cca16a38da6caa7d52151f7136895

• MPFR Fixes Patch - 17 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/mpfr-3.1.0-fixes-1.patch

MD5 sum: 6a1a0be6f2326e237ce27a0254e360a5

• Patch Testsuite Fix Patch - 1 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/patch-2.6.1-test_fix-1.patch

MD5 sum: c51e1a95bfc5310635d05081472c3534

• Perl Libc Patch - 1 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/perl-5.14.2-libc-1.patch

MD5 sum: 23682f20b6785e97f99d33be7719c9d6

• Perl Security Patch - 1 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/perl-5.14.2-security_fix-1.patch

MD5 sum: 7fa3e7e11fecf9d75f65452d700c3dd5

• Procps HZ Errors Patch - 2.3 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/procps-3.2.8-fix_HZ_errors-1.patch

MD5 sum: 2ea4c8e9a2c2a5a291ec63c92d7c6e3b

• Procps Watch Patch - 3.5 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/procps-3.2.8-watch_unicode-1.patch

MD5 sum: cd1a757e532d93662a7ed71da80e6b58

• Readline Upstream Fixes Patch - 1.3 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/readline-6.2-fixes-1.patch

MD5 sum: 3c185f7b76001d3d0af614f6f2cd5dfa

• Shadow nscd Patch - 1.1 KB:

Download: http://www.linuxfromscratch.org/patches/lfs/7.1/shadow-4.1.5-nscd-1.patch

MD5 sum: 6fd6a209c1aa623bad913fcff20b7d8e

پیوست ب) پیکربندی پرونده‌ی rc.site

```
# rc.site
# Optional parameters for boot scripts.

# Distro Information
# These values, if specified here, override the defaults
DISTRO="Danial GNU/Linux" # The distro name
DISTRO_CONTACT="dani.behzi@gmail.com" # Bug report address
DISTRO_MINI="DGL" # Short name used in filenames for distro config

# Define custom colors used in messages printed to the screen

# Please consult `man console_codes` for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles

# These values, if specified here, override the defaults
BRACKET="\\033[1;34m" # Blue
FAILURE="\\033[1;31m" # Red
INFO="\\033[1;36m"    # Cyan
NORMAL="\\033[0;39m"  # Grey
SUCCESS="\\033[1;32m" # Green
WARNING="\\033[1;33m" # Yellow

# Use a colored prefix
# These values, if specified here, override the defaults
BMPREFIX="     "
SUCCESS_PREFIX="${SUCCESS}  *  ${NORMAL}"
FAILURE_PREFIX="${FAILURE}*****${NORMAL}"
WARNING_PREFIX="${WARNING} *** ${NORMAL}"

# Interactive startup
IPROMPT="yes" # Whether to display the interactive boot promp
itime="3"    # The ammount of time (in seconds) to display the prompt

# The total length of the distro welcome string, without escape codes
wlen=$(echo "Welcome to ${DISTRO}" | wc -c )
welcome_message="Welcome to ${INFO}${DISTRO}${NORMAL}"

# The total length of the interactive string, without escape codes
ilen=$(echo "Press 'I' to enter interactive startup" | wc -c )
```

```
i_message="Press '${FAILURE}I${NORMAL}' to enter interactive startup"

# Set scripts to skip the file system check on reboot
#FASTBOOT=yes

# Skip reading from the console
#HEADLESS=yes

# Skip cleaning /tmp
#SKIPTMPCLEAN=yes

# For setclock
UTC=1
CLOCKPARAMS=

# For consolelog
#LOGLEVEL=5

# For network
HOSTNAME=Danial-Project

# Delay between TERM and KILL signals at shutdown
#KILLDELAY=3

# Optional sysklogd parameters
#SYSKLOGD_PARMS="-m 0"

# Console parameters
#UNICODE=1
#KEYMAP="de-latin1"
#KEYMAP_CORRECTIONS="euro2"
#FONT="lat0-16 -m 8859-15"
#LEGACY_CHARSET=
```

منابع و مآخذ

فهرست منابع فارسی

۱. انجمن‌های فارسی اوبونتو به آدرس http://forum.ubuntu.ir

فهرست منابع غیرفارسی

1. GNU website: http://gnu.org
2. Linux From scratch: http://linuxfromscratch.org
3. Stack Exchange: http://stackexchange.com
4. Linux Questions: http://linuxquestions.org
5. Ask Ubuntu: http://askubuntu.com

Abstract

In this report, first We will become familiar with different parts of an operating system. Featuring their free impleminations which are most choosen from GNU[1] project, It will be explained that how they should be put among together to form a working OS. Then these parts will be compiled with different configurations on demand to fit in their correct place to create the desired structure. In the end there is a way to boot the created operating system directly from disk and run it.

[1]GNU's Not Unix

ISLAMIC AZAD UNIVERSITY

North Tehran Branch

B.Sc. Thesis

On Computer Engineering

Research Title:

Design And Imp. Of A Free OS For x86 Computers Architecture

Advisor:

Engineer Ali Rezaie

Prepared By:

Danial Behzadi

Spring 2012