Make File

نوع کاربری: کاربردی

سطح کاربری: مقدماتی

نویسنده: شهرام روحانی

پست الکترونیکی: sh_roohani@yahoo.com

تاریخ: ۱۱/۰۳/۸٤

نسخه: ۰.۱

منبع: ---

3.....Makefile چیست؟
4.....کامپایلر
5.....یک مثال عملی
6.....Makefile
7.....make

Makefile چیست؟

می دانید که تنها زبان قابل درك برای کامپیوترها زبانی است معروف **Machine Code** یا زبان ماشین که از رشته پیوسته ای از اعداد تشکیل شده است. اگر یك فایل اجرایی باینری را در یك ویرایشگر هگز باز کنید، این رشته اعداد را معمولا می توانید به صورت اعداد دو رقمی پشت سر هم در مبنای **16** ببینید. اما برنامه نوشتن به این زبان برای انسان تقریبا کشنده است! فقط تصور کنید چند سال ممکن است طول بکشد تا یك تیم حرفه ای از برنامه نویسان، نرم افزاری در حد و اندازه **OpenOffice.org** را مستقیما به زبان ماشین بنویسند. تازه اگر اصلا چنین کاری ممکن باشد، اصلا منطقی نیست. چراکه به احتمال قریب به یقین در آخر به خوبی **OpenOffice.org** فعلی از کار در نخواهد آمد! پس چاره چیست؟

برنامه نویسان از زبانهایی استفاده می کنند که، نسبت به زبان ماشین، به زبان انسانی بسیار نزدیك تر هستند و به ویژه برخی از آنها با توجه به فلسفه ساختشان برای پیشبرد پروژه های نرم افزاری بزرگ خیلی مناسب هستند. به اینگونه زبانها اصطلاحا زبانهای سطح بالا می گوییم. اما حالا مشکل دیگری پیش می آید. کامپیوترها اینگونه زبانها را درك نمی کنند.

کامپایلر

کامپایلرها وظیفه تبدیل زبانهای سطح بالا و قابل درك برای انسان را به سطح پایین ترین زبان، یعنی زبان ماشین به عهده دارند. معمولا (تقریبا همیشه!) پروژه های نرم افزاری از بیش از یك فایل تشکیل شده اند که هر کدام بخشی از متن برنامه یا **Source Code** را در بر دارد. شما باید ابتدا هر فایل نوشته شده را جداگانه به وسیله کامپایلر به زبان ماشین تبدیل کنید. اصطلاحا به فایلهایی که در این مرحله تولید می شود، فایل **Object** می گویند. محتوای لین فایلها به زبان ماشین است، اما هنوز قابل اجرا توسط کامپیوتر نیستند! چرا؟ چون هر کدام از آن فایلها یا وابستگیهایی در فایلهای دیگر دارند که در این وضعیت برای سیستم غیر قابل تشخیص است، یا اساسا به تنهایی مفهومی ندارند که قابل اجرا باشند.مثلا ممکن است یکی از آنها حاوی مجموعه ای از دستورات (اصطلاحا تابع) باشد که روی ورودی خود پردازش خاصی انجام می دهند و نتیجه را برمی گردانند. اما این توابع زمانی مفید هستند که از جایی فراخوانی شوند و احتمالا فراخوانی تابع مورد نظر ما هم از داخل یکی از فایلهای **Object** دیگر صورت می گیرد.

در اینجاست که شما باید با کمك لینکر آن فایلهای **Object** را به ترتیب مناسب به هم پیوند دهید تا در نهایت به یك فایل اجرایی دست پیدا کنید.

یك مثال عملی

فرض کنید برنامه ای به زبان **C** در یك فایل به نام **main.c** نوشته اید و حالا می خواهید آن را به فایل قابل اجرا توسط کامپیوتر تبدیل کنید. این کار به سادگی و در دو خط دستور (که می شود آنها را در یك خط هم خلاصه کرد) قابل اجرا است:

$ gcc -c main.c

$ gcc -o main main.o

دستور اول از فایل **C** به نام **main.c** یك فایل **Object** به نام **main.o** می سازد و دستور دوم نیز از فایل **main.o** یك فایل اجرایی به نام **main** می سازد که حالا دیگر می توانید آن را اجرا کنید.

اما حالا فرض کنید که فایل **main.c** وابستگیهایی هم به دو فایل دیگر به نامهای **dep.c** و **dep.h** داشته باشد:

$ gcc -c main.c

$ gcc -c dep.c

$ gcc -o main main.o dep.o

می بینید که یك مرحله به مراحل کامپایل برنامه اضافه شد.حالا فرض کنید که به جای یك فایل یا سه فایل، این پروژه از دهها و صدها فایل تشکیل شده باشد. چند ساعت باید وقت صرف کنید تا پروژه را کامپایل کنید، تازه اگر همه چیز به خوبی پیش برود؟

به همه اینها اضافه کنید گزینه های خط دستور کامپایلر را که گاهی کار کامپایل از خط دستور را به فرایندی بسیار پیچیده تبدیل می کنند. حالا هربار که تغییراتی دربرنامه بدهید، باید همه کارها را از اول انجام دهید. یا به اندازه کافی فصفر بسوزانید و به خاطر داشته باشد که هر بار در کدام فایلها تغییر داده اید تا فقط آنها را کامپایل کنید و کمی وقت کمتری صرف کنید! آیا راه بهتری نیست؟

Makefile

چقدر خوب می شد اگر می توانستید همه این دستورات کامپایل را فقط یك بار تایپ کنید. خبر خوب! می توانید همه این دستورات را در فایل که به **Makefile** معروف است قرار دهید و برنامه ای به نام **make** را اجرا کنید تا آن برنامه خودش از روی **Makefile** برایتان بقیه کارها را انجام دهد و خروجی نهایی قابل اجرا را بسازد.

پس **Makefile** مجموعه دستورات لازم برای کامپایل یك پروژه نرم افزاری است که در قالبی خاص در یك فایل متنی ذخیره شده است.

به خصوص اگر شما برنامه نویس نباشید، برایتان قابل توجه خواهد بود که مجبور نیستید هربار اینهمه دستورات عجیب و غریب را تایپ کنید و تنها با تایپ دستور **make** زندگی برایتان آسان می شود.

make

همانطور که اشاره شد، **make** برنامه ای است که از روی **Makefile** کار کامپایل پروژه را انجام می دهد. در کل دو مزیت عمده می توان برای **make** در نظر گرفت:

1. همانطور که قبلا هم گفتم با استفاده از **Makefile** کارهای سخت (نوشتن دستورات پیچیده کامپایل) فقط یکبار انجام می شود.
2. **make** از روی برچسب زمانی فایلهای سورس و **Object** تشخیص می دهد که کدامیك از فایلها نیاز به دوباره کامپایل شدن دارند و به این ترتیب از کامپایل مجدد و غیر ضروری فایلهایی که از زمان آخرین کامپایل تغییری نکرده اند، جلوگیری می شود. چراکه فرایندکامپایل به ویژه در کامپیوترهای قدیمی و با پروژه های بزرگ کاری بسیار وقت گیر است.

make بعد از فراخوانی به ترتیب دنبال یك فایل با یکی از نامهای **makefile**، **GNUmakefile** یا **Makefile** می گردد تا دستورات کامپایل را از داخل آن خوانده و اجرا کند.