C/C++ Geliştirme Ortamları

Mühendis Köyü

2020-10-08

Contents

Bir Tutam Yazı

Önsöz

C/C++ geliştirme ortamı sadece bu diller ile kod yazmayı kapsamıyor. *IDE* (**Integrated Development Environment – Tümleşik Geliştirme Ortamı**) ortamı dışında geliştirdiğimiz yazılımlarda derleme, bağlama vb. işlemler sırasında projelere göre farklılık gösteren uzun paremetrelerle kullanmak durumunda kalırız. *IDE* kullanımı ise bizi *IDE*'ye bağımlı kıldığı gibi duruma göre işletim sistemine de bağımlı kılabilir. Bunun yanı sıra C/C++ projelerinde bu dillerde *dâhili* (**built-in**) olarak gelmeyen bir çok kütüphane kullanımı mevcuttur. Peki birden fazla farklı ortamda bu kütüphanelerin o ortamlara göre varlığı, nereden indirileceği gibi problemleri geliştiriciler elle mi gerçekleştirmek zorundadır?

İşte bu kitapla bahsi geçen problemler için geliştirilen çözümlere, çözümlerin oluşturduğu yeni problemlere getirilen çözümlere ve en son da hala devam etmekte olan veya daha da yeni olan problemlere değinmeyi Mühendis Köyü olarak amaç edindik.

Kapsamlı bir Türkçe kaynak olmasını hedefleyerek başladığımız bu yolculuğumuzda türü ne olursa olsun bizlere ulaşacak her bir eleştiri sönük bir mumun alev almasına yardımcı olan yanan bir mumun ateşi olacaktır.

Katkıda Bulunanlar

Genellikle kitabın en az bir bölümünü en fazla 1 kişi üstlenecek şekilde bir strateji belirledik. Bölüm başlarında sorumlu kişinin adı geçmektedir. Bununla birlikte kitap oluşturulurken emek vermiş kişilerinde burada geçmesini istedik.

Ahmet B. ÖZYURT	BARIŞ KIZILKAYA	Enes AYDIN
Erdem GÜNEŞ	Muhammed E. KOCAER	Numan F. AYDIN
Salih MARANGOZ	Semanur AYDINLIK	Senanur PAKSOY
Süleyman E. IŞIK		

Lisans

Bu Kitap Nasıl Geliştiriliyor

Mühendis Köyü telegram grubunda bulunan kişilerce gönüllülük esasına dayalı olarak bu kitaba girişilmiştir. Kitap, *R Markdown*'da *bookdown* paketi kullanılarak yazılmaktadır. Mühendis Köyü *Github* organizasyonu altında bulunan C-Cpp-gelistirme-ortamlari reposunun **master** *dalına* (**branch**) *CGOY* (**C Geliştirme Ortamı Yazarları**) ekibi tarafından yapılan değişiklikler yüklenmekte, yine ekipten biri tarafından **gh-pages** dalına ise *R Markdown* olarak yazılan projenin *HTML* çıktısı yüklenmektedir. Kitap geliştirilirken *Trello* üzerinden paylaşım, *telegram* üzerinden yardımlaşma, haberleşme ve tartışma sağlanmaktadır.

Part I

IDE

Giriş

Basit Geliştirme Ortamı (Editör + Derleyici)

Açıklama

Bu kısımda önsözde kısaca bahsettiğimiz problemlerin çıkış noktaları ile C/C++ için inşa sistemleri ve paket yöneticilerine değineceğiz.

Editör + Derleyici Kullanımı

Bir çok yazılım dilinde olduğu gibi programlarımızı IDE'ler olmadan da geliştirme imkanımız bulunmaktadır. IDE'ler güçlü bir geliştirme imkanı sunmasına rağmen yeni başlayanlar için karmaşık bir hal alabilmektedir. Aynı zamanda kaynak tüketimi editörlere göre daha yüksektir. Bu bölümde C yazılımları geliştirebileceğiniz en temel seviyeli editör olan **Vim**'e değinip, programlarınızı nasıl manuel olarak derleyebileceğinizden bahsedeceğiz.

Bu başlık altında VIM editörü hakkında anlatılacak bilgiler MIT | The Missing Semester of Your CS Education TR'den alınmıştır. VIM dahil diğer dersleri kaynak linkinden inceleyebilirsiniz.

VIM

Vim'in Felsefesi

Programlama yaparken zamanınızın çoğunu yazmaya değil okumaya/düzenlemeye harcarsınız. Bu yüzden Vim farklı modlara sahip bir editördür: metin eklemek veya metin işlemek için farklı modlara sahiptir. Vim programlanabilir (Vimscript ve Python gibi diğer diller ile) ve Vim'in arayüzünün kendisi bir programlama dilidir. Vim, fare kullanımından kaçınır, çünkü çok yavaştır; Vim, çok fazla hareket gerektirdiği için ok tuşlarını kullanmaktan bile kaçınır.

Sonuç olarak Vim, düşündüğünüz kadar hızlı olan bir editördür.

Modal Düzenleme Vim'in tasarımı, uzun metin akışları yazmak yerine, çok sayıda programcının zamanını; okumak, gezinmek ve küçük düzenlemeler yapmak için harcandığı fikrine dayanır. Bu nedenle Vim'in birden fazla çalışma modu vardır.

- **Normal:** dosyanın içerisinde gezinmek ve değişiklikler yapmak için,
- **Insert:** metin eklemek için,
- **Replace:** metni değiştirmek için,
- **Visual (Plain, Line or Block):** metin bloklarını seçmek için,
- **Command-line:** bir komut çalıştırmak için

Klayve tuşlarının farklı çalışma modlarında farklı anlamları vardır. Örnek olarak, **Insert** modunda iken `x` harfine basarsak o harfi ekleyecektir ama Normal modda iken `x` harfi imlecin altındaki karakteri siler ve Visual modda ise seçili olanı siler.

Varsayılan ayarlarda Vim, o anki çalışma modunu sol altta gösterir. Başlangıç modu/varsayılan mod Normal moddur. Genellikle zamanının çoğunu Normal mod ve Insert mod arasında geçireceksin. Herhangi bir moddan Normal moda geri dönmek için `<ESC>` tuşuna basarak modları değiştirebilirsiniz. Normal moddan `i` ile Insert moduna, `R` ile Replace moduna, `v` ile Visual moduna, `V` ile Visual Line moduna, `<C-v>` ile Visual Block moduna, `:` ile Command-line moduna girebilirsin.

Temel Öğeler

Metin Ekleme

Normal modda iken Insert moduna girmek için `i` tuşuna basın. Şimdi Vim, Normal moda geri dönmek için `<ESC>` tuşuna basana kadar diğer metin editörleri gibi çalışır. Bu bilgi ve yukarıda açıklanan temel bilgilerle birlikte, Vim'i kullanarak dosyaları düzenlemeye başlamak için ihtiyacınız olan tek şeydir (eğer bütün zamanınızı Insert Modundan düzenleme için harcıyorsanız çok da verimli değil).

Command-line

Command moduna Normal modda iken `:` yazarak giriş yapabiliriz. `:` Tuşuna bastığınızda bilgisayarınızın imleci ekranın altındaki komut satırına atlayacaktır. Bu mod, dosyaları açma, kaydetme, kapatma ve Vim'den çıkış yapma gibi birçok işleve sahiptir.

- `:q` çıkış (pencereyi kapatır)

- `:w` kayıt ("yaz")
- `:wq` kaydet ve çık
- `:e {dosyanın adı}` düzenlemek için dosyayı açar
- `:ls` açık bufferları gösterir
- `:help {konu}` yardımı açar
  - `:help :w` komutu için yardımı açar
  - `:help w` tuşu için yardımı açar

Dersin devamına ve konunun detayına Missing Semester TR kaynağından devam edebilirsiniz. Bunun yanı sıra eğer pratik yapmak istiyorsanız terminale `vimtutor` yazarak Vim ile birlikte gelen eğitimi tamamlayabilirsiniz.

VIM'de C Programı Geliştirme

`vim merhaba.c` ile merhaba isimli C dosyamızı açıp `i` harfi ile Insert moduna geçerek geleneksel uygulamamızı yazmaya başlayabiliriz.

```c
#include <stdio.h>
int main()
{
  printf("Merhaba Mühendis Köyü!");
  return 0;
}
```

Kodu tamamladıktan sonra `<ESC>` tuşu ile komut moduna geçip `:wq` yazarak kaydedip çıkış yapıyoruz. Yazdığımız kodu gcc ile derlemek için (clang kullanıyorsanız gcc yerine clang yazmanız yeterlidir):

```
gcc merhaba.c //derlenmiş kodu a.out olarak çıktı verir.
```

```
gcc merhaba.c -o merhaba //derlenmiş kodu merhaba olarak çıktı
verir.
```

Derlenen kodu çalıştırmak için `./a.out` veya `./merhaba` komutunu çalıştırmanız yeterlidir.

Editör kullanarak yazdığımız kodlarda, programımızı derlerken dahil ettiğimiz kütüphaneleri manuel olarak bağlantılamamız (kullandığımız kütüphaneyi derleyiciye belirtmemiz) gerekmektedir. Buna örnek bir kod örneği vermemiz gerekirse:

```
vim us_alma.c
```

```c
#include <stdio.h>
#include <math.h>
```

```c
int main()
{
    int taban, us;

    printf("Taban sayısını giriniz: ");
    scanf("%i", &taban);

    printf("Üs sayısını giriniz: ");
    scanf("%i", &us);

    printf("%1.f\n", pow((double)taban,(double)us));

    return 0;
}
```

Yazdığımız kodu kaydedip kapattıktan sonra derleme işlemini aşağıdaki şekilde gerçekleştirirseniz derleyici bilinmeyen referans hatası verecektir.

```
gcc us_alma.c -o us_alma
```

Math kütüphanesini dahil ettiğimiz kodumuzu belirtmiş olduğumuz şekilde ekli kütüphaneye bağlantı vererek derlemek için derleme komutuna -lm komutunu dahil etmemiz gerekmektedir. `-l` komutu bağlantılama (linkleme) komutu olup kendisinden sonra gelen argümandaki kütüphaneyi derleme işlemine dahil eder.

```
gcc us_alma.c -o us_alma -lm
```

Bu şekilde kodumuz uygun bir biçimde derlenecektir. `./us_alma` yazarak kodumuzu çalıştırabiliriz.

Vim ile temel seviyede C kodu yazılması ve derlenmesi bu şekilde özetlenebilir. Vim veya başka bir editör (VS Code, SublimeText vb.) kullanarak yazdığımız programları manuel olarak derlememiz gerekmektedir. Programlarımızın kapsamı genişledikçe bağlantılamamız gereken kütüphane sayısı artmakta ve bunu sürekli olarak yapmak zor bir hal almaktadır. Bu durumları hızlandırmak için ise makefile dediğimiz linkleme işlemlerini bizim için gerçekleştiren programların yazılmasına bu dokümanın ileriki bölümlerinde değineceğiz.

Part II

İnşa Sistemleri

Giriş

Makefile

Açıklama

Makefile kullanımı, GNU Make, BSD Make and NMake varyasyonlarına ve arasındaki farklara değinilecektir.

Kullanım

Makefile kullanımı, GNU Make, BSD Make and NMake varyasyonlarına ve arasındaki farklara değinilecektir.

Ayrıntılar

...

Avantajlar

Makefile kullanımı, GNU Make, BSD Make and NMake varyasyonlarına ve arasındaki farklara değinilecektir.

Dezavantajlar

Makefile kullanımı, GNU Make, BSD Make and NMake varyasyonlarına ve arasındaki farklara değinilecektir.

Cmake

Açıklama

Çoğu programlama dili ile yazılan kodun çalıştırılabilir olması için derlenmesi gerekir.Derleme işlemi sırasında derleyiciye uygun parametrelerin eklenmesi, gerekli kütüphanelerin dahil edilmesi, birim testlerin çalıştırılması gibi işlemler yapılır. Tüm bu işlemler için sadece IDE kullanılabildiği gibi IDE ile birlikte başka araçlarlar ile de yapılabilir. Ancak IDE programlarının ayarları birbirinden farklıdır (Eclipse, VS). Bir IDE tüm işletim sistemlerinde yer almayabilir (VS). Aynı IDE farklı işletim sistemlerinde farklı ayarlara sahip olabilir (Eclipse). Önceki bölümde görüldüğü üzere Makefile ile bu iş yapılaiblir ancak Makefile'ın gerektirdiği make sadece Linux/Unix tabanlı işletim sistemlerinde gayet iyi çalışmaktadır. Windows'da fazla bir kullanım söz konusu değildir (nmake aracı olsa da). Bu durumları görmezden gelsek bile proje büyüdüğünde Makefile ile uğraşmak zorlayıcı olabilmektedir. Öte yandan, bir makefile ile yönetilen projeyi bağımsız şekilde istenilen bir IDE ortamında açmak uğraştırıcı olmaktadır.

İşte bu gibi sorunlara Cmake yardımcı oluyor. CMake C/C++ program oluşturma sürecini işletim sisteminden ve derleyiciden bağımsız bir şekilde gerçekleştirmemize olanak sağlayan açık kaynak kodlu ve çapraz platformlu bir sistemdir. Sisteminizdeki derleyici ve IDE'ye göre projeyi kullanabilecek konfigürasyonları üretebilir. Ek olarak test etme ve paketleme yapmamızı da sağlar.

- Cmake Kullanan Bazı Projeler
  - OpenCV Bilgisayar görüsü alanında en popüler açık kaynak bir kütüphane.
  - KDE KDE'nin masaüstü programı
  - MySQL Dünyanın en bilinen açık kaynaklı veri tabanı programı
  - Boost C++ için standart birçok kütüphane içeren oldukça popüler kütüphaneler kümesi.

Kullanım

Öncelikle, "https://cmake.org/download" sayfasından işletim sisteminize göre kurulum yapılabilir. Alternatif olarak, ilgili ortamlar için tercih edilebilecek linkler:

- https://chocolatey.org/packages/cmake
- https://packages.msys2.org/base/mingw-w64-cmake

Linux ortamında ise aşağıdaki formatta kurabilirsiniz.

```
$ <package_manager dnf, apt-get> install cmake
```

Nasıl ki, make uygulaması için bir Makefile gerekliyse Cmake uygulaması içinde bir dosya gerekli, ismi de "CmakeLists.txt". Bu dosyanın içine, proje adı, çalıştırılabilir dosyalar, kütüphanelerin yönetimi (nasıl ekleneceği) gibi ayarlar yaparak aslında proje ile alakalı daha soyutsal bilgiler veririz. Sonra cmake programı zaten hiyerarşiyi, proje ismini, bağımlı olunan kütüphaneleri vb. bilgileri içeren dosyanın yardımıyla istediğimiz IDE için varsa yine istediğimiz derleyici için konfügurasyonlar oluşturacağız hatta ve hatta Makefile'ın kendinisini de bu yolla oluşturabileceğiz.

Şimdi basit bir Cmake projesi yapalım. Öncelikle bir "app.cpp" oluşturalım içine "Merhaba Dünya" örneği kod yazalım.

```cpp
#include <iostream>

int main()
{
  std::cout << "Hello world cmake" << '\n';
  return 0;
}
```

"CMakeLists.txt" dosyasının oluşturulalım ve içine aşağıdaki yazıları yazalım.

```
project(TestApp)
cmake_minimum_required(VERSION 3.16.3)
add_executable(TestApp app.cpp)
set(TestApp_VERSION_MAJOR 1)
set(TestApp_VERSION_MINOR 0)
```

- Bu dosya içerisinde aşağıdaki temel ayarların anlamına değinelim.
  - project : Projenin adını belirttiğimiz yerdir. Eğer cmake kullanarak VS için proje dosyaları üretirsek "TestApp.sln" dosyası oluşacaktır.

  - cmake_minimum_required : Proje için gerekli olan en düşük Cmake sürümününü belirtir. Cmake sürümleri arasında farklılıklar vardır. Cmake geliştikçe yeni özellikler kazanmaktadır.
  - add_executable : Çalıştırılabilir dosyanın hangi dosyalarla oluşturulacağını ifade eder. Tabi büyük projelerde dosyaları el ile bu şekilde yazmıyoruz.
  - set : Bir değişkene (sol taraf) bir değer atar (sağ taraf). Buradaki değişkenler Cmake tarafından kullanılacak global değişkenler olabildiği gibi, kendinize özgü de olabilir.

Evet, bu kadar. Başka birşeye gerek yok. Şimdi elimizde Cmake ile desteklenmiş bir proje var. Nasıl kullanacağız?

Ayrıntılar

- Dahil edilecek dosyaların aranacağı yolları eklemek için (include path):

  ```
  include_directories(include_dir)
  ```

- Kütüphane eklemek için:

  ```
  add_library(Cfile STATIC FxParser) : statik kütüphane
  add_library(Cfile SHARED FxParser) : paylaşılan kütüphane
  add_library(Cfile MODULE FxParser) : modül kütüphane
  ```

- `Static` yalnızca derleme zamanında doğrudan gömülebilen kütüphanedir.
- `Shared` derleme zamanında linklenebilen ve çalışma zamanında yüklenebilen kütüphanedir.
- `Module` derleme zamanında linkleme olmaksızın, çalışma zamanında ihtiyaç hâlinde yüklenebilen kütüphanedir.

CMakeLists.txt dosyamıza çeşitli seçenekler ekleyip bunları kullanabiliriz. Örnek olarak doxygen ile dokümantasyon oluşturulması ve testlerin derlenmesi seçenekler olsun.

Seçenekler için syntax şu şekildedir:

```
option(ayar_adı "ayarın açıklaması" varsayılan_durum[ON/OFF])
```

```
option(BUILD_DOXYGEN "Insa dokumentasyonu" OFF)
option(BUILD_TESTS "Insa testleri" OFF)
```

- Alt Proje Dizini Oluşturma

Oluşturulan kütüphaneyi kullanmak için `add_subdirectory` ile kütüphane için oluşturulan CMakeLists.txt projeye eklenir.

```
add_subdirectory(kutuphaneAdı)
```

- Mantıksal Operatörler

CMakeLists.txt dosyasına mantıksal operatörler gibi alt dizinler ekleyebiliriz.

```
if(BUILD_TESTS)
  add_subdirectory(test)
endif(BUILD_TESTS)

if(BUILD_DOXYGEN)
  add_subdirectory(docs)
endif(BUILD_DOXYGEN)
```

- Değişken Atama

CMake ile değişken atamak için set komutunu kullanmamız yeterlidir. Bu komut ile hem CMake'in global değişkenlerini atayabiliriz hem de yeni değişkenler oluşturup değer atayabiliriz.

```
set : versiyon atama
set(Cfile_VERSION_MINOR 0) -> versiyon atama
set(disable_derivative on) -> değişken oluşturma & değer atama
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/bin) -> global değişkene değ
```

Tüm anlatılanları toplama açısından bir CMakeLists.txt örneği aşağıdadır.

```
project(Cfile)

cmake_minimum_required(VERSION 3.16.3)

option(BUILD_DOXYGEN "Insa dokumentasyonu" OFF)
option(BUILD_TESTS "Insa testleri" OFF)

set(disable_derivative on)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/bin)
```

```
add_subdirectory(kutuphaneAdı)

if(BUILD_TESTS)
    add_subdirectory(test)
endif(BUILD_TESTS)

if(BUILD_DOXYGEN)
    add_subdirectory(docs)
endif(BUILD_DOXYGEN)
```

- Bazı Önemli Cmake Değişkenleri
  - CMAKE_SOURCE_DIR -> CMakeLists.txt dosyasının bulunduğu üst seviye klasör.
  - PROJECT_NAME -> Proje adı. project() komutuyla belirlenebilir.
  - PROJECT_SOURCE_DIR -> Proje kaynak klasörünün tam yolu.
  - CMAKE_MAJOR_VERSION -> Majör Cmake sürüm numarası
  - CMAKE_MINOR_VERSION -> Minör Cmake sürüm numarası
- Derleyiciye Parametre Eklemek

```
set(CMAKE_CXX_FLAGS "Hello World" -> derleyicinin tek parametresi
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} Hello World") -> mevcut parametrelerin sonuna ekleme ya
```

- Include Klasörü Eklemek

```
include_directories("inc")
```

  Kütüphane proje eklendikten sonra `target_link_libraries` ile kütüphane projeye bağlanır.

- Kütüphaneleri Bağlamak

```
target_link_libraries(Cfile ${Boost_LIBRARIES})
```

- Örnek olması açısından Ubuntu terminal üzerinden Cmake kullanarak "Hello World" yazdıralım.

```
    ~/Desktop
mkdir helloworld   -> helloworld klasörü oluşturduk.
~/Desktop
cd helloworld
~/Desktop/helloworld
vim main.cpp
```

- main.cpp şu şekildedir.

```
#include <iostream>
using namespace std;
int main() {
     cout << "Hello World!" << endl;
     return 0;
}
```

```
~/Desktop/helloworld
vim CMakeLists.txt
#CMakeLists.txt şu şekildedir.
cmake_minimum_required(VERSION 3.16.3)
project(helloworld)
add_executable(
    helloworld
     main.cpp
)
```

```
~/Desktop/helloworld
cmake .  -> cmake derliyoruz.
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/saturn/Desktop/helloworld
~/Desktop/helloworld
make
Scanning dependencies of target helloworld
[ 50%] Building CXX object CMakeFiles/helloworld.dir/main.cpp.o
[100%] Linking CXX executable helloworld
[100%] Built target helloworld
~/Desktop/helloworld
```

```
  ./helloworld -> dosyamızı derleyip çalıştırıyoruz.
Hello World!
```

Avantajlar

- Verimlidir.
  - Uzun süreli kod yazma, düşük süreli inşa sistemi çözme.
  - Her proje için açık kaynaklı ve ücretsizdir.
- Güçlü bir yapısı vardır.
  - Cmake çoklu geliştirme ortamlarını destekler ve aynı projede derler.
  - C/C++/CUDA/Fortran/Python gibi birçok programlama dilini destekler.
  - Cmake ile üçüncü parti kütüphaneleri projelerinize entegre edebilirsiniz.
- Karmaşık işleri basite indirger.
- Kolay öğrenilir.
- Çapraz platformdur.
- Evrenseldir.
- Makefile dosyalarını Cmake yazar.
- Organizasyonu sağlar.

Dezavantajlar

- Dili/Söz dizimi
  - Cmake dili önceden kullandığınız dillerle karşılaştırılmamalıdır. Çünkü sınıf(class), eşleme(map) ve sanal bir fonksiyon ya da lambda ifadeleri içermez. Yeni başlayanlar için giriş argümanlarını çözümlemek ve bir fonksiyonun sonuç döndürmesini anlamak karmaşıktır.
- İş akışına etkisi
  - Cmake kullanırken projenin inşa konfigürasyonunu doğrudan IDE üzerinden güncelleştirilemez. CMakeLists.txt dosyasına eklenen ya da değiştirilen her target dosyası yazılmalı. Aksi halde IDE üzerinden yapılan güncellemeler ve değişiklikler Cmake çalıştırıldığında kaybolacaktır.
  - Cmake'de inşa betik dilini(script) çalıştırmak için en az bir araç gereklidir.
  - Sınırlı dokümantasyonu vardır.

GNU Autotools

Açıklama

GNU İnşa Sistemi (**GNU Build System**) olarak da bilinen GNU Autotools, kaynak kod ile program kurulumlarını ve programların birçok *Unix benzeri* (**Unix-like**) sisteme taşınabilir hale getirilmesine yardımcı olmak için tasarlanmış bir programlama araçları bütünüdür. Autotools, *GNU araç zincirinin* (**GNU toolchain**) bir parçasıdır ve birçok özgür yazılım ve açık kaynak projelerinde yaygın olarak kullanılmaktadır. GNU İnşa Sistemi çoğunlukla Unix benzeri veya *POSIX benzeri* (**POSIX-like**) işletim sistemlerinde, özellikle Linux'ta ve Gömülü Linux projelerinde kullanılır. Linux, MacOSX, FreeBSD, OpenBSD, NetBSD vb. işletim sistemlerinde kullanılabilir. Windows ortamında iyi bir şekilde çalışmamaktadır. GNU İnşa Sistemi kullanılarak birçok programlama dili (C, C++, Objective C, Fortran, Fortran77, Erlang) ile program paketleri oluşturabilirsiniz.

- GNU Autotools Kullanan Bazı Projeler
  - GNU Emacs Oldukça popüler gelişmiş metin düzenleyici ve daha fazlası...
  - Glibc GNU C Standart Kütüphanesi
  - GCC Çeşitli programlama dilleri için bir derleyici sistemidir. GNU araç zincirinin bir parçasıdır.
  - NetworkManager Linux ağ yöneticisi arka plan programı
  - Nhttp2 HTTP 2'nin C'de uyarlanması
- Destekleyen bazı IDE'ler
  - Eclipse
    - Linux Araçları Eklentisi
    - Eclipse CDT Autotools Kullanım Kılavuzu
  - NetBeans
    - CppGnuAutoTools Eklentisi
  - QTCreator

∗ Autotools Proje Yöneticisi Eklentisi

GNU İnşa Sisteminin kullanıldığı projelerde şu dosyalar bulunmalıdır : “configure.ac” ve “Makefine.am”. Genellikle bu projeleri yüklerken şu komut dizisi çalıştırılır : ./configure && make && make install.

Kullanım

Basit bir GNU Autotools projesi ile kullanımı göstermiş olacağız. Detaylı bilgiler ve ileri seviye kullanım için yazının sonunda paylaşılan belgelere bakabilirsiniz.

Öncelikle GNU Autotools için iki aracın Linux sisteminize yüklü olması gerekmektedir. Bunlar : autoconf automake. Eğer bunlar sisteminizde yoksa genellikle aşağıdaki formatta kurabilirsiniz.

```
$ <package_manager dnf, apt-get> install autoconf automake
```

Sisteminizde kullandığınız dil ile alakalı araçların bulunduğunu varsayıyoruz.

- Adımlara geçmeden önce ne yapacağımız ile ilgili açıklayıcı maddeleri verelim.
  - Basit bir C++ projesi için bir tane kaynak dosyası oluşturacağız ve içine ekrana “Merhaba Dünya” çıktısını verecek kodu yazacağız. “NEWS, INSTALL, README, COPYING, AUTHORS, ChangeLog” dosyalarını oluşturacağız. GNU kodlama standartlarına göre bu dosyalarının da bu dizinde bulunması gereklidir. GNU bu konuda baya bir katıdır. Genelde bir proje yaparken bu gibi dosyaları geçin kaynak kod içerisine bile birşeyler yazmayı tercih etmeyebiliyor geliştiriciler.
  - “Makefile.am” dosyası oluşturacağız. Bu dosyayı automake aracı kullanacak. Bu dosya proje içindeki klasör yapısını, derlenecek programlar için bilgileri, projemiz için oluşturacağımız paket içeriğinde olmasını istediğimiz başlık dosyalarını, man sayfalarını, veri (resim, video vb.) dosyalarını gibi bilgileri içerebilir. İşlem sonunda “Makefile.in” dosyası oluşur.
  - “configure.ac” dosyası oluşturacağız. Bu dosyayı autoconf aracı kullanacak. “configure.ac” yerine “configure.in” dosyası da olabilirdi. Lakin yeni paketler için artık önerilen “configure.ac” dosya ismidir. Bu dosyaya bu autoconf'un anlayabileceği yazı kuralında (Bash betik diline benzer) proje ile alakalı bilgileri yazacağız. Bu bilgiler derleyici bilgisi, bazı kütüphanelerin ve kütüphane yollarınının bulunması, bazı başlık dosyalarının varlığının kontrolü gibi ayarlamalar olabilir. Örneğin “AC_PROG_CC” isimli autoconf makrosunu bu dosyada kullandığımızda autoconf bu dosyayı okurken CC isimli değişkene

C derleyicisini atayacaktır. “AC_PROG_CXX” de C++ için kullanılır. İşlem sonunda “configure” programı oluşur.

C/C++ projelerinde derleme aşamasında derleyiciye -D parametresi ile bazı makro değerleri verilerek derleme aşamasının yönlendirilmesi sağlanabilmektedir. Bunun ise burada çözümü “config.h” kullanmaktır. “autoheader” programı “configure.ac” içeriğine bakarak ilgili derleyici parametre ayarlamaları (başlık dosyasını kullanmak için) yapmaktadır.

GNU Autotools araçları birbirinden bağımsız araçlar değildir ve 2 tane de değillerdir (autoheader, aclocal, autoconf ve automake). Tek tek de biz çağırmayacağız zaten, bizim yerimize bir komut (autoreconf) ile işlemler kendi başına doğru sırayla çalışacak. İkinci ve üçüncü aşama için daha detaylı açıklamaları yukarıda bahsettiğim gibi yazının sonundaki linklerden bulabilirsiniz. Kısaca açıklamak gerekirse, autoconf programı “configure.ac” içindeki eski bir şablon dili olan M4 makrolarını okuyarak shell programı üretir. “configure” programının amacı sisteme özgü “config.h” ve Makefile dosyalarını oluşturmaktır. “automake” ise “Makefile.am” içindeki mantıksal öbeklere göre “Makefile.in” dosyasını (dizinler varsa dosyalarını) oluşturur. “automake” programının amacı ise “configure” programı için gerekli “Makefile.in” dosyasını oluşturmaktır.

- Geliştirici :
  - “Makefile.am” ve “configure.ac” dosyalarını yazar.
  - “autoreconf –install” komutunu çalıştırır. (Alternatif yöntemlerde var.)
    - “Makefile.am” -> “Makefile.in”
    - “configure.ac” -> “configure”
- Kullanıcı :
  - ./configure
    - “config.h” ve Makefile
  - make && make install.
    - Çıktı

A. C++ Projesi ve Gerekli Dosyalar

“test_app” klasörü altında “app.cpp” isimli bir C++ kaynak dosyası oluşturalım ve içine aşağıdaki kodu kopyalayalım. Dosya ismi : main.cpp

```
#include <iostream>

int main()
```

```
{
  std::cout << "Hello world autotools" << '\n';
  return 0;
}
```

"NEWS, INSTALL, README, COPYING, AUTHORS, ChangeLog" dosyalarını oluşturalım.

```
$ touch NEWS INSTALL README COPYING AUTHORS ChangeLog
```

"configure.ac" dosyasına eğer AM_INIT_AUTOMAKE([foreign]) eklenirse, "foreign" den ötürü bu dosyaların şartı kalkacaktır. Lakin bunu es geçiyoruz.

B. "Makefile.am" dosyası

Kaynak kodları genelde birden fazla dizinlerde tutarız Ancak her bir dizin için "Makefile" ile uğraşmamız gerekecektir. Buna yönelik çözüm olan "mantıksal bir dil" tarzında "Makefile.am" dosyaları oluşturuyoruz.

```
bin_PROGRAMS = TestApp
TestApp_SOURCES = app.cpp
```

"bin_PROGRAMS = TestApp" sanki bir atama işlemi gibi gelebilir ama aslında burada 3 farklı bilgi mevcut. "bin_", "PROGRAMS", "TestApp". Açıklaması ise şu, oluşacak program (PROGRAMS) "TestApp" isminde bin klasöründe oluşsun. "bin_" öneki yerine "lib, include, data" gibi tanımlı değerler kullanabiliriz. "PROGRAMS" yerine ise 'LIBRARIES', 'LTLIBRARIES', 'LISP', 'PYTHON', 'JAVA', 'SCRIPTS', 'DATA', 'HEADERS', 'MANS', ve 'TEXINFOS' hedef türlerinden birini yazabiliriz. Şimdi ise bir gereklilik var. TestApp dedik ama TestApp hangi kaynak kodlara bağlı yazmadık. İşte "PROGRAMS" ile hedef türünü belirttiğimiz isimlerin "_SOURCES" ile bağlı olduğu kaynak kodları vermemiz gerekmektedir. Yani "TestApp_SOURCES = app.cpp" bu kadar.

TestApp eğer bir kütüphaneye bağımlı ise örneğin "TestApp_LDADD = -lm" diyerek linker için bilgi ekleyebiliriz. "PROGRAMS" için belirtilen isim için "_SOURCES" ve "_LDADD" yanında "_LDFLAGS" ve "_DEPENDENCIES" tanımlamarı da vardır. Burada sadece bahsederek geçiyoruz.

C. "configure.ac" dosyası

"Makefile.am" dosyasında ne derleyiciyi ayarı vardı ne de kurulum ile alakalı adımlar. İşte bu dosyanın amacı da bu. Öncelikle ne yazacağımıza, sonra da açıklamalarına bakalım.

```
AC_INIT( [TestApp], [0.1], [email@email.com])
AC_PREREQ([2.69])
AM_INIT_AUTOMAKE([-Wall -Wextra])
AC_CONFIG_FILES([Makefile])
AC_PROG_CXX
AC_OUTPUT
```

- AC_INIT : AC_INIT makrosu, 2'si zorunlu olmak üzere 5 argümana sahiptir. "configure" programı için ayarlamalar yapar, "configure" çalıştırılırken gönderilecek parametreleri işler. Zorunlu ilk iki parametre Paket İsmi ile Versiyon bilgisidir.
- AC_PREREQ : "autoconf" programının olması gereken minumum versiyonunu parametre olarak alır.
- AM_INIT_AUTOMAKE : "automake" programına parametreler yollar.
- AC_CONFIG_FILES : "configure" programının oluşturacağı dosya parametre olarak geçilir. "config.h" başlık dosyası oluştursaydı AC_CONFIG_HEADERS([config.h]) eklenmeliydi.
- AC_PROG_CXX : Sistemdeki C++ derleyicisini bulur.
- AC_OUTPUT : Bu makro kullanılmalıdır. "config.status" isminde bir kabul program oluşturup çalıştıracaktır. Bunun amacı "config.h" ve Makefile dosyalarını oluşumu içindir.

Evet şimdi sadece bu komutu çalıştırmak yetecektir:

```
$ autoreconf --install
```

İşlemlerimiz bitti şimdi aşağıdaki işlemleri uygulayarak hem test etmiş olacağız hem de bir tarball oluşturmuş olacağız. "make distcheck" komutu bir tar.gz dosyası oluşturacaktır.

- Genel ./configure kullanım örneği.

```
$ ./configure
$ make
$ make distcheck
```

Bu arada programımızın çıktısını ./TestApp diyerek gözlemleyebilirsiniz. "configure" programınıza aşağıdaki gibi parametre yollayabilirsiniz.

```
$ ./configure CC=clang CXX=clang++ --prefix=$HOME/opt/test_app
```

Ayrıntılar

...

- Belgeler
  - Autoconf ve Automake Kullanımı
  - Autotools Gizemini Çözüyoruz
  - autoconf, automake Kullanımı
  - CPP / C++ - Building Systems and Build Automation
  - How To Use Autotools
  - GNU Autoconf
  - GNU Automake
  - GNU Libtool
  - GNU Autotools Kitabı
  - Embedded Linux Conference 2016 - GNU Autotools Tutorial
  - Chapter 1: A brief introduction to the GNU Autotools
  - Autotools - Fedora Developers
  - C/C++ Project Built with GNU Build System (A.K.A. GNU Autotools): NetBeans vs. Eclipse CDT
- Terminal Belgeleri:
  - `$ info automake`
  - `$ info autoconf`
  - `$ info libtool`

Avantajlar

GNU Autotools bir nevi kendi derleme sistemini oluşturarak, bir yazılımın her hangi bir sistemde 2-3 komut ile çalışabilir hale gelmesini sağlayabilir.

Dezavantajlar

GNU Autotools yapımcıları tarafından kullanımı basit şeklinde ifade ediliyor ama kullanımı biraz uğraştırıcıdır.

Msbuild

Açıklama

Msbuild kullanımı avantajı dezavantajı...

Kullanım

Msbuild kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Msbuild kullanımı avantajı dezavantajı...

Dezavantajlar

Msbuild kullanımı avantajı dezavantajı...

Meson

Açıklama

- Meson, yazılımın oluşturulmasını (derlenmesini) otomatikleştirmek için bir yazılım aracıdır. Meson'un genel amacı programcı verimliliğini arttırmaktır. Meson, Apache 2.0 Lisansı altında Python ile yazılmış ücretsiz ve açık kaynaklı bir yazılımdır.

- Meson'un belirtilen bir diğer amacı, modern programlama araçları ve en iyi uygulamalar için birinci sınıf destek sağlamaktı. Bunlar, birim testi, kod kapsamı raporlaması, önceden derlenmiş başlıklar ve benzerleri gibi çeşitli özellikleri içerir. Bu özelliklerin tümü Meson kullanan herhangi bir projeye anında ulaşılabilir olmalıdır. Kullanıcının bu özellikleri alabilmesi için üçüncü taraf makroları araması veya kabuk komut dosyaları yazması gerekmez.Meson bunları kullanıcıların yerine yapar.

–> Meson, MacOS,Windows ve diğer işletim sistemleri de dahil olmak üzere Unix benzeri işletim sistemlerinde yerel olarak çalışır.

–> Meson, C, C++, CUDA, D, Objective-C, Fortran, Java, C#, Rust ve Vala dillerini destekler.

–> Wrap adı verilen bağımlılıkları işlemek için bir mekanizmaya sahiptir.

–> Meson, Ninja, GNU derleyici Koleksiyonu, Clang, Microsoft Visual Studio ve isteğe bağlı olarak diğer derleme sistemlerini destekler.

–> Çok okunabilir ve kullanıcı dostu Turing olmayan eksiksiz bir DSL'de tanımlar oluşturur.

–> Meson, Meson ve CMake alt projelerini destekler. Bir Meson derleme dosyası WrapDB hizmetine de başvurabilir.

Kullanım

Meson, kullanımı olabildiğince basit olacak şekilde tasarlanmıştır. Meson, Python 3'te uygulanmaktadır ve 3.5 veya daha yenisini gerektirir. İşletim sisteminiz bir paket yöneticisi sağlıyorsa, onunla birlikte yüklemelisiniz. Paket yöneticisi olmayan platformlar için, Python'un ana sayfasından indirmeniz gerekir.

0.0.1 Meson Yükleme

Meson sürümleri GitHub sürüm sayfasından indirilebilir ve özel bir şey yapmadan `./meson.py`'yi bir sürümden veya git deposundan çalıştırabilirsiniz.

Windows'ta, Python'u Python komut dosyalarını çalıştırılabilir yapan yükleyici seçenekleriyle kurmadıysanız, `python /path/to/meson.py` çalıştırmanız gerekir; burada `python` Python 3.5 veya daha yeni sürümüdür.

En yeni geliştirme kodunu doğrudan Git'ten alabilirsiniz.

0.0.2 Gereksinimler

- Python 3
- Ninja

0.0.3 Bağımlılıklar

En yaygın durumda, Meson'da varsayılan olan `ninja` arka ucunu kullanmak için Ninja yürütülebilir dosyasına ihtiyacınız olacaktır. Bu arka uç, GCC, Clang, Visual Studio, Mingw, ICC, ARMCC vb.dahil olmak üzere tüm platformlarda ve tüm toolchains zincirlerinde kullanılabilir.

Mümkünse paket yöneticiniz tarafından sağlanan sürümü kullanabilirsiniz, aksi takdirde çalıştırılabilir ikili dosyayı Ninja projesinin yayın sayfasından indirebilirsiniz.

Windows'ta Visual Studio çözümleri oluşturmak için yalnızca Visual Studio arka ucunu (`--backend = vs`) veya macOS'ta XCode projeleri oluşturmak için XCode arka ucunu (`--backend = xcode`) kullanmalısınız, Ninja'ya ihtiyacınız yoktur.

0.0.4 Meson Pip İle Kurulum

Meson, Python Paket Dizininde mevcuttur ve kök gerektiren ve sistem genelinde yükleyecek olan `pip3 install meson` ile kurulabilir.

Alternatif olarak(önerilir), kullanıcınız için kuracak ve herhangi bir özel ayrıcalık gerektirmeyen `pip3 install --user meson`'u kullanabilirsiniz. Bu, paketi `~/.local/` dizinine yükleyecektir, dolayısıyla `path`'inize `~/.local/bin` eklemeniz gerekecektir.

0.0.5 Paket Yöneticisi Kullanarak Kurulum

Ubuntu:

```
$ sudo apt-get install python3 python3-pip python3-setuptools \
                       python3-wheel ninja-build
```

! Distro paketli yazılım hızla modası geçmiş olabilir.

0.0.6 Sorun Giderme

Ortak sorunlar:

```
$ meson builddir
$ bash: /usr/bin/meson: No such file or directory
```

Açıklama: Python pip modülü kurulumu için varsayılan kurulum ön eki, kabuk ortamınız path içerisine dahil değildir.Python pip kurulum modülleri için varsayılan ön ek `/ usr / local` altında bulunur.

** Çözüm: Bu sorun, varsayılan kabuk ortamı path'i `/ usr / local / bin`içerecek şekilde değiştirilerek çözülebilir.

Not: Bu sorunu çözmenin sembolik bağları kullanmak veya ikili dosyaları varsayılan bir yola kopyalamak gibi başka yolları da vardır. Bu yöntemler, paket yönetiminin birlikte çalışabilirliğini bozabileceği için önerilmez veya desteklenmez.

0.0.7 Meson Projesi Derleme

Meson'un en yaygın kullanım durumu, üzerinde çalıştığınız bir kod tabanında kod derlemektir. Atılacak adımlar çok basit.

```
$ cd /path/to/source/root
$ meson builddir && cd builddir
$ meson compile
$ meson test
```

Unutulmaması gereken tek şey, ayrı bir yapı dizini oluşturmanız gerektiğidir. Meson, kaynak ağacınızın içinde kaynak kodu oluşturmanıza izin vermez. Tüm yapı eserleri, yapı dizininde saklanır. Bu, aynı anda farklı konfigürasyonlara(yapılandırmalara) sahip birden fazla yapı ağacına sahip olmanızı sağlar. Bu şekilde oluşturulan dosyalar kazara revizyon kontrolüne eklenmez.

Kod değişikliklerinden sonra yeniden derlemek için `meson compile` yazmanız yeterlidir. Build (yapı) komutu her zaman aynıdır. Kaynak kodunda rastgele değişiklikler yapabilir ve sistem dosyalarını oluşturabilirsiniz . Meson bunları algılar ve doğru olanı yapar. Optimize edilmiş ikili dosyalar oluşturmak istiyorsanız, Meson'u çalıştırırken `--buildtype = debugoptimized` argümanını kullanın. Optimize edilmemiş yapılar için bir yapı dizini ve optimize edilmiş yapılar için de bir tane yapı dizini tutmanız önerilir. Herhangi bir yapılandırmayı derlemek için, ilgili yapı dizinine gidin ve `meson compile`'i çalıştırın.

Meson, hata ayıklama bilgilerini ve derleyici uyarılarını (yani `-g` ve `-Wall`) etkinleştirmek için otomatik olarak derleyici bayrakları ekleyecektir.Bu, kullanıcının onlarla uğraşmak zorunda olmadığı ve bunun yerine kodlamaya odaklanabileceği anlamına gelir.

0.0.8 Meson'u Dağıtım Paketleyici Olarak Kullanma

Dağıtım paketleyicileri genellikle kullanılan derleme bayrakları üzerinde tam kontrol isterler. Meson bu kullanım durumunu yerel olarak desteklemektedir. Meson projelerini oluşturmak ve kurmak için gereken komutlar aşağıdadır:

```
$ cd /path/to/source/root
$ meson --prefix /usr --buildtype=plain builddir -Dc_args=... -Dcpp_args=... -Dc_link_a
$ meson compile -C builddir
$ meson test -C builddir
$ DESTDIR=/path/to/staging/root meson install -C builddir
```

Komut satırı anahtarı `--buildtype=plain` Meson'a komut satırına kendi bayraklarını eklememesini söyler. Bu, paketleyiciye kullanılan işaretler üzerinde tam kontrol sağlar.

Bu, diğer yapı sistemlerine çok benzer.Tek fark, `DESTDIR` değişkeninin `meson kurulumuna` bir argüman olarak değil, bir ortam değişkeni olarak geçirilmesidir.

Dağıtım derlemeleri her zaman sıfırdan gerçekleştiği için, daha hızlı olduklarından ve daha iyi kod ürettiklerinden paketlerinizin üzerinde unity oluşturmayı etkinleştirmeyi düşünebilirsiniz. Bununla birlikte, unity yapıları etkinleştirilmiş olarak oluşturulmayan birçok proje vardır, bu nedenle unity yapılarını kullanma kararı, paketleyici tarafından duruma göre yapılmalıdır.

0.0.9 Include Kullanımı

Çoğu C / C ++ projesinin kaynaklardan farklı dizinlerde başlıkları vardır. Bu nedenle, içerme dizinlerini belirtmeniz gerekir. Bir alt dizinde olduğumuzu ve bunun `include` alt dizinini bazı hedefin arama yoluna eklemek istediğimizi varsayalım. Bir dahil etme dizini nesnesi oluşturmak için şunu yapıyoruz:

```
incdir = include_directories('include')
```

`İncdir` değişkeni artık `include` alt dizine bir başvuru tutar. Şimdi bunu bir yapı hedefine argüman olarak aktarıyoruz:

```
executable('someprog', 'someprog.c', include_directories : incdir)
```

Bu iki komutun herhangi bir alt dizinde verilebileceğini ve yine de çalışacağını unutmayın. Meson, konumları takip edecek ve hepsinin çalışması için uygun derleyici bayrakları oluşturacaktır.

Unutulmaması gereken bir diğer nokta da `include_directories`'in hem kaynak dizini hem de ilgili yapı dizinini path'ı içerecek şekilde eklemesidir, bu yüzden dikkat etmeniz gerekmez.

0.0.10 Hello World Yazımı

Önce kaynağı tutan bir main.c dosyası oluşturuyoruz. Şuna benziyor:

```
#include<stdio.h>

int main(int argc, char **argv) {
  printf("Hello world!\n");
  return 0;
}
```

Daha sonra bir Meson build (yapı) açıklaması oluşturup aynı dizindeki `meson.build` adlı bir dosyaya koyuyoruz. İçeriği aşağıdaki gibidir:

```
project('tutorial', 'c')
executable('demo', 'main.c')
```

Hepsi bu. Autotools'tan farklı olarak, kaynaklar listesine herhangi bir kaynak başlığı eklemenize gerek olmadığını unutmayınız.

Artık uygulamamızı oluşturmaya hazırız. Öncelikle kaynak dizine girip aşağıdaki komutu yazarak yapıyı başlatmamız gerekiyor:

```
$ meson builddir
```

Tüm derleyici çıktısını tutmak için ayrı bir yapı dizini oluşturuyoruz. Meson, kaynak içi derlemelere izin vermediği için diğer bazı derleme sistemlerinden farklıdır. Her zaman ayrı bir yapı dizini oluşturmanız gerekir. Genel kural, varsayılan yapı dizinini en üst düzey kaynak dizininizin bir alt dizinine koymaktır.

Meson çalıştırıldığında aşağıdaki çıktıyı yazdırır:

```
The Meson build system
 version: 0.13.0-research
Source dir: /home/jpakkane/mesontutorial
Build dir: /home/jpakkane/mesontutorial/builddir
Build type: native build
Project name is "tutorial".
Using native c compiler "ccache cc". (gcc 4.8.2)
Creating build target "demo" with 1 files.
```

Artık kodumuzu oluşturmaya hazırız.

```
$ cd builddir
```

```
$ meson compile
```

Bunu yaptıktan sonra ortaya çıkan ikiliyi çalıştırabiliriz.

```
$ ./demo
```

Bu beklenen çıktıyı üretir.

```
Hello world!
```

Ayrıntılar

...

Avantajlar

Meson kullanımının avantajları:

1-Meson'a başlamak kolaydır.

2-Meson, CMake ile karşılaştırılabilir ve kendi kriterlerine göre diğer build (yapı) sistemlere kıyasla en hızlısıdır.

3-Cmake'e kıyasla sadece zaman kazandırmaz. Aynı zamanda daha net ayırma, daha iyi bir iş akışı, okunabilir seçenekler, dosyalar oluşturma,önceden derlenmiş başlıklar ve Unity gibi daha fazla hedef bulundurur. Ayrıca, Meson bu kullanım durumu için baştan tasarlandığından, diğer dillerle karıştırmak daha kolaydır. Alt projeler ve çapraz derleme tamamen desteklenir ve sonradan düşünülmüş gibi hissetirmez.

4-Meson dili güçlü* bir şekilde yazılmıştır, öyle ki kütüphane, yürütülebilir, dize ve bunların listeleri gibi yerleşik türler birbirinin yerine kullanılamaz. Özellikle, Make'dan farklı olarak, liste türü boşluktaki dizeleri ayırmaz. Böylece, dosya adlarındaki ve program argümanlarındaki boşluk ve diğer karakterler temiz bir şekilde işlenir. (*Güçlü bir şekilde yazılan bir dilin derleme zamanında zayıf yazıma göre daha katı yazma kuralları vardır.)

5-Çoklu platform özelliği vardır.

6-Birçok desteklenen dil bulunmaktadır.

7-Ağaç dışı yapıya sahiptir.

8-x86_64 Unix'te doğru kütüphane kurulum dizinini ayarlama otomatiktir.

9-Pkg-config dosya üreticisi vardır.

Dezavantajlar

Meson kullanımının dezavantajları:

1-Meson, projeleri kaynaktan entegre etmek için kendi indirme hizmetini destekliyor, ancak şu anda çok fazla paketi yok.

2-Statik bağlantı desteğinin olmaması ve (işlevlerin olmaması nedeniyle) genişletilecek herhangi bir yeteneğin olmaması meson için bir dezavantajdır.

3-Pkg-config olmadan kütüphane bağımlılıkları bulamamaktadır.

4-Özel fonksiyonlar ile genişletilemez.

5-Hata ayıklama yapıları varsayılan olarak en iyi duruma getirilmemiştir.

–» Avantaj ve dezavantaj bölümünde bulunan bazı maddeler slant sitesinde bulunan kullanıcı yorumlarıdır.

Qmake

Açıklama

Qmake kullanımı avantajı dezavantajı...

Kullanım

Qmake kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Qmake kullanımı avantajı dezavantajı...

Dezavantajlar

Qmake kullanımı avantajı dezavantajı...

Kbuild

Açıklama

Kbuild kullanımı avantajı dezavantajı...

Kullanım

Kbuild kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Kbuild kullanımı avantajı dezavantajı...

Dezavantajlar

Kbuild kullanımı avantajı dezavantajı...

Build2

Açıklama

Build2 kullanımı avantajı dezavantajı...

Kullanım

Build2 kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Build2 kullanımı avantajı dezavantajı...

Dezavantajlar

Build2 kullanımı avantajı dezavantajı...

Xmake

Açıklama

Xmake kullanımı avantajı dezavantajı...

Kullanım

Xmake kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Xmake kullanımı avantajı dezavantajı...

Dezavantajlar

Xmake kullanımı avantajı dezavantajı...

Bazel

Açıklama

Bazel kullanımı avantajı dezavantajı...

Kullanım

Bazel kullanımı avantajı dezavantajı...

Ayrıntılar

...

Avantajlar

Bazel kullanımı avantajı dezavantajı...

Dezavantajlar

Bazel kullanımı avantajı dezavantajı...

Part III

Paket Yöneticileri

Giriş

Conan

Conan Bölüm içeriği

Conan kullanımı avantajı dezavantajı...

Part IV

XXXXX XXXXX

Giriş

Docker

Açıklama

Conan kullanımı avantajı dezavantajı...