Insper

12 - Suporte para projetos

SuperComputação - 2018/2

Igor Montagner, Luciano Soares

Na última aula houve muitas dúvidas sobre a parte tecnológica do projeto, em especial a utilização do CMake para gerenciar o projeto e o uso de ferramentas de debug. Este roteiro descreve o básico da utilização de CMake e da configuração e utilização de um debugger em conjunto com uma IDE (KDevelop) ou com um editor de textos voltado a programação (VSCode).

Atenção: Usar ferramentas adequadas aumenta consideravelmente a produtividade e também facilita encontrar erros no código. A partir desta aula será obrigatório já ter debugado o programa antes de tirar dúvidas.

Parte 1 - CMake

O CMake é atualmente a ferramenta mais usada para gerenciar projetos C/C++. Um projeto é definido por um arquivo nomeado *CMakeLists.txt* onde é definido nome do projeto, quais dependências são usadas e o quais arquivos serão gerados pelo projeto. Cada arquivo gerado é chamado de *target* na nomenclatura usada pelo CMake e pode ser um executável ou uma biblioteca (estática ou dinâmica).

Um arquivo *CMakeLists.txt* básico pode conter apenas três linhas (supondo que você tenha um arquivo *hello_world.cpp* que seja um hello world em C++, se não tiver crie :)

```
cmake_minimum_required(version 3.9)
project (projeto_basico)
add_executable(hello hello_world.cpp)
```

Este arquivo somente descreve o projeto. Para efetivamente compilarmos o programa precisamos passar pela fase de *configuração*, em que o *CMake* checa se todas as dependências foram encontradas e se os compiladores necessários estão instalados. Se tudo estiver em ordem podemos gerar um *Makefile* (para Linux) ou um projeto do Visual Studio (para Windows).

Para fazer a configuração do projeto basta rodar o comando `cmake` mais o caminho para a pasta do projeto. Você só precisará refazer a configuração do projeto se modificar o arquivo *CMakeLists.txt*. É boa prática fazer a compilação do código em uma pasta separada, como na sequência de comandos abaixo.

```
mkdir build
cd build
cmake ..
```

Estes comandos devem ter gerado uma série de arquivos na pasta *build*, incluindo um *Makefile*. Para compilar o projeto basta rodar

```
make
```

E um executável de nome *hello* deverá aparecer na pasta *build*.

Exercício 1: dependências

Em grande parte dos projetos usamos bibliotecas externas e o CMake nos ajuda a garantir que elas estão instaladas e que a versão é compatível com a do nosso projeto. Dizemos quais bibliotecas precisamos usando o comando `find_package`.

- `find_package(nome)` - procura pela biblioteca *nome*
- `find_package(nome COMPONENTS a b c)` - procura pelos componentes *a,b,c* da biblioteca *nome*. Usado para bibliotecas enormes com partes opcionais.

Cada biblioteca define quais variáveis do CMake são definidas. Após a biblioteca ser encontrada precisamos dizer para o CMake onde estão os arquivos de cabeçalho (variáveis nomeadas com algo parecido com `*_INCLUDE_DIR`) e as bibliotecas dinâmicas que usaremos (variáveis nomeadas com algo parecido com `*_LIBRARY_DIR`). Fazemos isto usando os comandos `include_directories` e `link_directories` em conjunto com as variáveis exportadas por cada biblioteca.

Exercício: vamos criar um arquivo *CMakeLists.txt* para o arquivo *find_the.cpp* disponibilizado nesta aula. O nome do *target* criado deverá ser *find_the*.

Como usamos a biblioteca *boost-filesystem*, precisamos consultar como usar *Boost* em conjunto com *CMake*. Isto está disponível na documentação oficial do CMake.

Você deverá:

1. indicar onde estão os includes do Boost
2. indicar onde encontrar as bibliotecas do Boost
3. indicar que o target *find_the* utiliza a biblioteca boost-filesystem.

Exercício 2: usando OpenMP

Vamos agora adicionar ao nosso projeto um segundo executável, definido no arquivo *teste_omp.cpp* e que chamaremos de *teste_omp*. Para compilá-lo precisamos setar as flags do OpenMP, mas o faremos somente para este target.

1. Você pode usar `find_package` para encontrar o OpenMP. Pesquise como fazê-lo e quais variáveis são exportadas.
2. Assim como no *Boost*, são definidos *targets* pré-prontos para usarmos diretamente nas dependências do nosso *target teste_omp*. Encontre o nome desse *target* na documentação e use-o.

Exercício 3: seu projeto em uma IDE

Na última parte iremos abrir nosso projeto um ambiente de desenvolvimento em vez de continuar trabalhando na linha de comando. Serão providenciados links para tutoriais para o VSCode e para o KDevelop.

1. Abrir o projeto na IDE de sua escolha.
2. Compilar os dois executáveis.
3. Criar configurações para rodar cada um deles.

VSCode

O VSCode não vem com suporte por padrão a projetos CMake, mas a extensão *Cmake Tools* (de vector-of-bool) possui um bom suporte. Instale-a e carregue o projeto. A extensão possui uma documentação tanto para a etapa de configuração e compilação quanto para execução dos programas e debugging.

Para executar os programas você pode usar a opção "Run in terminal" disponível no menu do botão direito de cada *target*.

KDevelop

O KDevelop já possui suporte a projetos CMake por padrão. Basta abrir o projeto usando o menu principal ("Project -> Open/Import project") e apontar para uma pasta contendo um arquivo *CMakeLists.txt*.

Você precisará criar um *Launch Configuration* para cada executável. A documentação neste link mostra um guia de como fazê-lo. Para projetos usando CMake ao invés de escrever o nome do executável na mão (como no link acima) podemos selecionar o nome do *target* no campo *Project target*.

Escolhemos qual é o programa a ser executado no menu "Run -> Current Launch Configuration".

Parte 2 - Ferramentas de debug

Nesta seção veremos como usar o *gdb* em conjunto com alguma IDE. Estamos supondo que você já configurou sua IDE com CMake no passo anterior e é capaz de rodar cada executável de maneira independente.

Se você rodou o programa *find_the* notou que a saída dele está incorreta. Por alguma razão ele dá print em todas as linhas e não somente nas que a palavra *the* ocorre. Vamos usar ferramentas de Debug no VSCode ou no KDevelop para entender o problema.

Para iniciar, vamos colocar um *breakpoint* na linha 23 do arquivo *find_the.cpp*. Ao rodar o código no debugger a execução será interrompida na linha 23 e poderemos explorar o estado atual do programa (valores das variáveis) e rodar o código linha a linha.

- **Continue**: roda até o próximo *breakpoint*
- **Step Over**: roda a linha atual e passa para a próxima.
- **Step Into**: se a linha atual tem uma chamada de função, continua o debug dentro da função chamada.
- **Step Out**: executa até o fim da função atual e para logo após o retorno.

Vamos usar estes comandos em uma sessão guiada de debug para localizar o problema.

1. Inicie o debug. O programa vai para na linha 23 e esperar um comando. Use "Step Over" para ir para a próxima linha.
2. Se usarmos "Step Over" de novo a função `busca_no_arquivo` será executada inteira, mas o que queremos é debugar ela também. Use "Step Into" para continuar o debug dentro da função.
3. Já dentro da função, use "Step Over" para chegar até a linha em que fazemos `temp.find(...)`. Veja no explorador de variáveis o valor de temp.
4. "Step Over" novamente. O programa entrou no `if` mesmo `temp` não contendo *the*! O problema deve estar na linha acima então!

Conserte este problema, remova todos os breakpoints e prossiga. Rode o código de novo e desta vez teremos um novo problema: as linhas impressas são sequenciais...

Exercício: execute o código com debug e pare sua execução na linha 12 (a que contém `temp.find(...)`). Examine o valor das variáveis no explorador de variáveis e, se necessário, rode o código passo a passo. Você consegue identificar o problema? Se sim, conserte-o. Se não, tire dúvida com um colega que já acabou a atividade ou chame o professor.

VSCode

Breakpoint: clique ao lado do número da linha. Um círculo vermelho indica que o breakpoint está ativo.

Iniciar Debug: botão direito no target -> "Run with debugger"

Comandos de Debug: presentes no topo da tela na ordem que foram apresentados acima.

Explorador de variáveis: painel no lado esquerdo, dentro da view de Debug.

KDevelop

Breakpoint: clique com o botão direito na linha e depois em "Toggle breakpoint" (Ctrl+Alt+B). A linha mudará de cor.

Iniciar Debug: botão *Debug* na barra de ferramentas ou "Run -> Debug Launch" (atalho F9)

Comandos de Debug: disponíveis na toolbar de debug e também apresentados na ordem acima.

Explorador de variáveis: disponível na barra lateral e nomeado como "Variables".

O KDevelop suporta várias customizações de visual, diferente do VSCode que já vem "pronto" e não permite muitas customizações.