O Programa Weaver

Thiago Leucz Astrizi

thiago@bitbitbit.com.br

Abstract: *This article describes using literary programming the program Weaver. This program is a project manager for the Weaver Game Engine. If a user wants to create a new game with the Weaver Game Engine, they use this program to create the directory structure for a new game project. They also use this program to add new source files and shader files to a game project. And to update a project with a more recent Weaver version installed in the computer. The presenting code in C is cross-platform and should work under Windows, Linux, OpenBSD and possibly other Unix variants.*

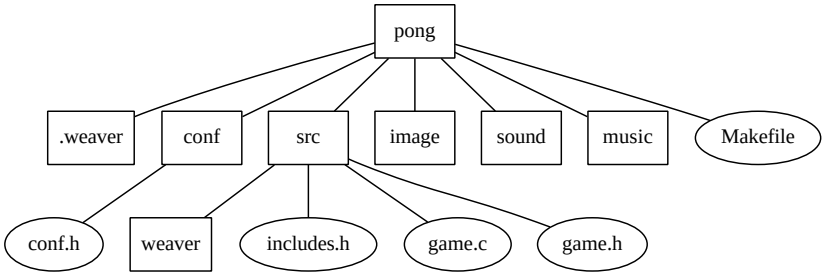
Resumo: *Este artigo descreve usando programação literária o programa Weaver. Este programa é um gerenciador de projetos para o Motor de Jogos Weaver. Se alguém deseja criar um novo projeto com o motor de jogos, usará este programa para criar a estrutura de diretórios desejada. Também usará o programa para adicionar novos arquivos de código-fonte e shaders. Para atualizar um projeto pré-existente com uma nova versão de Weaver, o programa também é necessário. O código seguinte em C será multi-plataforma e deverá funcionar em Windows, Linux, OpenBSD e possivelmente outras variantes de Unix.*

1. Introdução

Um motor de jogos é formado por um conjunto de bibliotecas e funções que auxiliam na criação de jogos fornecendo as funcionalidades mais comus para este tipo de desenvolvimento. Mas além das bibliotecas e funções, deve existir um gerenciador responsável por fazer com que o seu código utilize as bibliotecas a maneira adequada e faça as inicializações necessárias.

O motor de jogos Weaver tem pré-requisitos bastante estritos de como o diretório que contém um projeto Weaver deve estar organizado. É para cumprir erstes requisitos que o programa que será apresentado é necessário. Ele inicializa da maneira correta a estrutura de diretórios de um novo projeto. Ele adiciona novos arquivos fonte já com quaisquer código necessário para sua integração. E por controlar o projeto desta forma, ele saberá atualizar as bibliotecas para versões mais recentes se necessário.

O uso deste programa será por mieo de linha de comando. Por exemplo, se um usuário usar o comando "`weaver pong`", será criada uma estrutura de diretórios semelhante à mostrada na imagem que ilustra o fim da seção com um novo projeto chamado "pong".



As seguintes seções do artigo estão organizadas da seguinte forma. A seção 2 abordará a licensa do software. A seção 3 listará as variáveis usadas para controlar seu comportamento. A seção 4 trará algumas macros que usaremos, algumas das quais apareceram na estrutura do programa. A seção 5 apresentará algumas funções auxiliares que utilizaremos. A seção 6 mostrará a inicialização das variáveis do programa. A seção 7 mostrará os casos de uso do programa e como implementá-los após termos as variáveis com os valores certos.

2. Copyright e licenciamento

Segue abaixo a licença do programa e sua tradução não-oficial:

```
Copyright (c) Thiago Leucz Astrizi 2015

This program is free software: you can redistribute it and/or
modify it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public
License along with this program.  If not, see
<http://www.gnu.org/licenses/>.
```

```
Copyright (c) Thiago Leucz Astrizi 2015

Este programa é um software livre; você pode redistribuí-lo e/ou
modificá-lo dentro dos termos da Licença Pública Geral GNU Affero como
publicada pela Fundação do Software Livre (FSF); na versão 3 da
Licença, ou (na sua opinião) qualquer versão.

Este programa é distribuído na esperança de que possa ser útil,
mas SEM NENHUMA GARANTIA; sem uma garantia implícita de ADEQUAÇÃO
a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a
Licença Pública Geral GNU Affero para maiores detalhes.

Você deve ter recebido uma cópia da Licença Pública Geral GNU Affero
junto com este programa. Se não, veja
```

<http://www.gnu.org/licenses/>.

A versão completa da licença pode ser obtida junto ao código-fonte Weaver ou consultada no link mencionado.

Variáveis e Estrutura do Programa Weaver

O comportamento de Weaver deve depender das seguintes variáveis:

`inside_weaver_directory` : Indicará se o programa está sendo invocado de dentro de um projeto Weaver.

`argument` : O primeiro argumento, ou NULL se ele não existir

`argument2` : O segundo argumento, ou NULL se não existir.

`project_version_major` : Se estamos em um projeto Weaver, qual o maior número da versão do Weaver usada para gerar o projeto. Exemplo: se a versão for 0.5, o número maior é 0. Em versões de teste, o valor é sempre 0.

`project_version_minor` : Se estamos em um projeto Weaver, o valor do menor número da versão do Weaver usada para gerar o projeto. Exemplo, se a versão for 0.5, o número menor é 5. Em versões de teste o valor é sempre 0.

`weaver_version_major` : O número maior da versão do Weaver sendo usada no momento.

`weaver_version_minor` : O número menor da versão do Weaver sendo usada no momento.

`arg_is_path` : Se o primeiro argumento é ou não um caminho absoluto ou relativo para um projeto Weaver.

`arg_is_valid_project` : Se o argumento passado seria válido como nome de projeto Weaver.

`arg_is_valid_module` : Se o argumento passado seria válido como um novo módulo no projeto Weaver atual.

`arg_is_valid_plugin` : Se o segundo argumento existe e se ele é um nome válido para um novo plugin.

`arg_is_valid_function` : Se o segundo argumento existe e se ele seria um nome válido para um loop principal e também para um arquivo.

`project_path` : Se estamos dentro de um diretório de projeto Weaver, qual o caminho para a sua base (onde há o Makefile)

`have_arg` : Se o programa é invocado com argumento.

`shared_dir` : Deverá armazenar o caminho para o diretório onde estão os arquivos compartilhados da instalação de Weaver. Por padrão, será igual à "`/usr/local/share/weaver`", mas caso exista a variável de ambiente `WEAVER_DIR`, então este será considerado o endereço dos arquivos compartilhados.

`author_name` , `project_name` e `year` : Conterão respectivamente o nome do usuário que está invocando Weaver, o nome do projeto atual (se estivermos no diretório de um) e o ano atual. Isso será importante para gerar as mensagens de Copyright em novos projetos Weaver.

`return_value` : Que valor o programa deve retornar caso o programa seja interrompido no momento atual.

A estrutura geral do programa com a declaração de todas as variáveis será:

Arquivo: `src/weaver.c`**:**

<Seção a ser Inserida: Cabeçalhos Incluídos no Programa Weaver>
<Seção a ser Inserida: Macros do Programa Weaver>
<Seção a ser Inserida: Funções auxiliares Weaver>

```
int main(int argc, char **argv){
  int return_value = 0; /* Valor de retorno. */
  bool inside_weaver_directory = false, arg_is_path = false,
    arg_is_valid_project = false, arg_is_valid_module = false,
    have_arg = false, arg_is_valid_plugin = false,
    arg_is_valid_function = false; /* Variáveis booleanas. */
  unsigned int project_version_major = 0, project_version_minor = 0,
    weaver_version_major = 0, weaver_version_minor = 0,
    year = 0;
  /* Strings UTF-8: */
  char *argument = NULL, *project_path = NULL, *shared_dir = NULL,
    *author_name = NULL, *project_name = NULL, *argument2 = NULL;
                    <Seção a ser Inserida: Inicialização>
     <Seção a ser Inserida: Caso de uso 1: Imprimir ajuda (criar projeto)>
    <Seção a ser Inserida: Caso de uso 2: Imprimir ajuda de gerenciamento>
            <Seção a ser Inserida: Caso de uso 3: Mostrar versão>
         <Seção a ser Inserida: Caso de uso 4: Atualizar projeto Weaver>
           <Seção a ser Inserida: Caso de uso 5: Criar novo módulo>
           <Seção a ser Inserida: Caso de uso 6: Criar novo projeto>
            <Seção a ser Inserida: Caso de uso 7: Criar novo plugin>
           <Seção a ser Inserida: Caso de uso 8: Criar novo shader>
         <Seção a ser Inserida: Caso de uso 9: Criar novo loop principal>
END_OF_PROGRAM:
                      <Seção a ser Inserida: Finalização>
  return return_value;
}
```

4. Macros e Cabeçalhos do Programa Weaver

O programa precisará de algumas macros. A primeira delas deverá conter uma string com a versão do programa. A versão pode ser formada só por letras (no caso de versões de teste) ou por um número seguido de um ponto e de outro número (sem espaços) no caso de uma versão final do programa.

Para a segunda macro, observe que na estrutura geral do programa vista acima existe um rótulo chamado `END_OF_PROGRAM` logo na parte de finalização. Uma das formas de chegarmos lá é por meio da execução normal do programa, caso nada dê errado. Entretanto, no caso de um erro, nós podemos também chegar lá por meio de um desvio incondicional após imprimirmos a mensagem de erro e ajustarmos o valor de retorno do programa. A responsabilidade de fazer isso será da segunda macro.

Por outro lado, podemos também querer encerrar o programa previamente, mas sem que tenha havido um erro. A responsabilidade disso é da terceira macro que definimos.

Seção: Macros do Programa Weaver:

```
#define VERSION "Alpha"
#define ERROR() {perror(NULL); return_value = 1; goto END_OF_PROGRAM;}
#define END() goto END_OF_PROGRAM;
```

Como estamos usando a função de biblioteca `perror`, devemos incluir o cabeçalho `stdio.h`, o que também nos trará s funções de imprimir na tela, abrir e fechar arquivos e escrever neles, o que nos será útil. Vamos inserir suporte à valores booleanos que usamos na própria estrutura do programa e também a biblioteca padrão, que tem funções como `exit` usadas na estrutura do programa:

Seção: Cabeçalhos Incluídos no Programa Weaver:

```
#include <stdio.h> // printf, fprintf, fopen, fclose, fgets, fgetc, perror
#include <stdbool.h> // bool, true, false
```

```
#include <stdlib.h> // free, exit, getenv
```

5. Funções Auxiliares

Listemos aqui algumas funções que usaremos ao longo do programa para facilitar sua descrição.

5.1. path_up: Manipula Caminho

Para manipularmos o caminho da árvore de diretórios, usaremos uma função auxiliar que recebe como entrada uma string com um caminho na árvore de diretórios e apaga todos os últimos caracteres até apagar dois "/". Assim em "/home/alice/projeto/diretorio/" ele retornaria "/home/alice/projeto" efetivamente subindo um nível na árvore de diretórios.

É importante lembrar que no Windows o separador não é o "/", mas o """. Então vamos tratar o separador de forma diferente de acordo com o sistema:

Seção: Funções auxiliares Weaver:

```
void path_up(char *path){
#if !defined(_WIN32)
  char separator = '/';
#else
  char separator = '\';
#endif
  int erased = 0;
  char *p = path;
  while(*p != '\0') p ++; // Vai até o fim
  while(erased < 2 && p != path){
    p --;
    if(*p == separator) erased ++;
    *p = '\0'; // Apaga
  }
}
```

Note que caso a função receba uma string que não possua dois "/" em seu nome, acabamos apagando toda a string. Neste programa limitaremos o uso desta função a strings com caminhos de arquivos que não estão na raíz e diretórios diferentes da própria raíz que terminam sempre com "/", então não teremos problemas pois a restrição do número de barras será cumprida. Ex: "/etc/" e "/tmp/file.txt".

5.2. directory_exists: Arquivo existe e é diretório

Para checar se o diretório `.weaver` existe, definimos `directory_exist(x)` como uma função que recebe uma string correspondente à localização de um arquivo e que deve retornar 1 se `x` for um diretório existente, -1 se `x` for um arquivo existente e 0 caso contrário. Primeiro criamos as macros para não nos esquecermos do que significa cada número de retorno:

Seção: Macros do Programa Weaver (continuação):

```
#define NAO_EXISTE             0
#define EXISTE_E_EH_DIRETORIO  1
#define EXISTE_E_EH_ARQUIVO   -1
```

Seção: Funções auxiliares Weaver (continuação):

```
int directory_exist(char *dir){
#if !defined(_WIN32)
  // Unix:
  struct stat s; // Armazena status se um diretório existe ou não.
  int err; // Checagem de erros
```

```
  err = stat(dir, &s); // .weaver existe?
  if(err == -1) return NAO_EXISTE;
  if(S_ISDIR(s.st_mode)) return EXISTE_E_EH_DIRETORIO;
  return EXISTE_E_EH_ARQUIVO;
#else
  // Windows:
  DWORD dwAttrib = GetFileAttributes(dir);
  if(dwAttrib == INVALID_FILE_ATTRIBUTES) return NAO_EXISTE;
  if(!(dwAttrib & FILE_ATTRIBUTE_DIRECTORY)) return EXISTE_E_EH_ARQUIVO;
  else return EXISTE_E_EH_DIRETORIO
#endif
}
```

Dependendo de estarmos no Windows ou em sistemas Unix, usamos funções diferentes e vamos precisar de cabeçalhos diferentes:

Seção: Cabeçalhos Incluídos no Programa Weaver:

```
#if !defined(_WIN32)
#include <sys/types.h> // stat, getuid, getpwuid, mkdir
#include <sys/stat.h> // stat, mkdir
#else
#include <windows.h> // GetFileAttributes, ...
#endif
```

5.3. concatenate: Concatena strings

A última função auxiliar da qual precisaremos é uma função para concatenar strings. Ela deve receber um número arbitrário de srings como argumento, mas a última string deve ser uma string vazia. E irá retornar a concatenação de todas as strings passadas como argumento.

A função irá alocar sempre uma nova string, a qual deverá ser desalocada antes do programa terminar. Como exemplo, `concatenate("tes", " ", "te", "")` retorna `"tes te"`.

Seção: Funções auxiliares Weaver (continuação):

```
char *concatenate(char *string, ...){
  va_list arguments;
  char *new_string, *current_string = string;
  size_t current_size = strlen(string) + 1;
  char *realloc_return;
  va_start(arguments, string);
  new_string = (char *) malloc(current_size);
  if(new_string == NULL) return NULL;
   // Copia primeira string de acordo com o indicado pelo sistema operacional
#ifdef __OpenBSD__
  strlcpy(new_string, string, current_size);
#else
  strcpy(new_string, string);
#endif
  while(current_string[0] != '\0'){ // Pára quando copiamos o ""
    current_string = va_arg(arguments, char *);
    current_size += strlen(current_string);
    realloc_return = (char *) realloc(new_string, current_size);
    if(realloc_return == NULL){
      free(new_string);
      return NULL;
```

```
    }
    new_string = realloc_return;
     // Copia próxima string de acordo com o recomendado pelo sistema
#ifdef __OpenBSD__
    strlcat(new_string, current_string, current_size);
#else
    strcat(new_string, current_string);
#endif
  }
  return new_string;
}
```

É importante lembrarmos que a função `concatenate` sempre deve receber como último argumento uma string vazia ou teremos um *buffer overflow*. Esta função é perigosa e deve ser usada sempre tomando-se este cuidado.

O uso desta função requer que usemos o seguinte cabeçalho:

Seção: Cabeçalhos Incluídos no Programa Weaver:

```
#include <string.h> // strcmp, strcat, strcpy, strncmp
#include <stdarg.h> // va_start, va_arg
```

5.4. basename: Obtém o caminho do diretório de arquivo

Esta função já existe em sistemas Unix. Dado o caminho completo para um arquivo, ela retorna uma string apenas com o nome do arquivo. Ela não precisa alocar uma nova string, ela retorna um ponteiro para o nome do arquivo dentro do próprio caminho recebido como argumento. Vamos defini-la agora para sistemas Windows:

Seção: Funções auxiliares Weaver (continuação):

```
#if defined(_WIN32)
char *basename(char *path){
  char *p = path;
  char *last_delimiter = NULL;
  while(*p != '\0'){
    if(*p == '\')
      last_delimiter = p;
    p ++;
  }
  if(last_delimiter != NULL)
    return last_delimiter + 1;
  else
    return path;
}
#endif
```

Mesmo que não precisemos definir esta função em sistemas Unix, ainda precisamos incluí-la com o cabeçalho:

Seção: Cabeçalhos Incluídos no Programa Weaver:

```
#include <libgen.h>
```

5.5. copy_single_file: Copia um único arquivo para diretório

A função `copy_single_file` tenta copiar o arquivo cujo caminho é o primeiro argumento para o diretório cujo caminho é o segundo argumento. Se ela conseguir, retorna 1 e retorna 0 caso contrário.

Seção: Funções auxiliares Weaver (continuação):

```
int copy_single_file(char *file, char *directory){
  int block_size, bytes_read;
  char *buffer, *file_dst;
  FILE *orig, *dst;
  // Inicializa 'block_size':
  <Seção a ser Inserida: Descobre tamanho do bloco do sistema de arquivos>
  buffer = (char *) malloc(block_size); // Aloca buffer de cópia
  if(buffer == NULL) return 0;
  file_dst = concatenate(directory, "/", basename(file), "");
  if(file_dst == NULL) return 0;
  orig = fopen(file, "r"); // Abre arquivo de origem
  if(orig == NULL){
    free(buffer);
    free(file_dst);
    return 0;
  }
  dst = fopen(file_dst, "w"); // Abre arquivo de destino
  if(dst == NULL){
    fclose(orig);
    free(buffer);
    free(file_dst);
    return 0;
  }
  while((bytes_read = fread(buffer, 1, block_size, orig)) > 0){
    fwrite(buffer, 1, bytes_read, dst); // Copia origem -> buffer -> destino
  }
  fclose(orig);
  fclose(dst);
  free(file_dst);
  free(buffer);
  return 1;
}
```

O mais eficiente é que o buffer usado para copiar arquivos tenha o mesmo tamanho do bloco do sistema de arquivos. Para obter o valor correto deste tamanho, usamos o seguinte trecho de código em sistemas Unix:

Seção: Descobre tamanho do bloco do sistema de arquivos:

```
#if !defined(_WIN32)
{
  struct stat s;
  stat(directory, &s);
  block_size = s.st_blksize;
  if(block_size <= 0){
    block_size = 4096;
  }
}
#endif
```

No Windows o código equivalente seria:

Seção: Descobre tamanho do bloco do sistema de arquivos (continuação):

```
#if defined(_WIN32)
{
  DWORD dummy;
  DISK_GEOMETRY s;
```

```
  file = CreateFileW(".temp.dat", 0, FILE_SHARE_READ | FILE_SHARE_WRITE,
                     NULL, OPEN_EXISTING, FILE_FLAG_DELETE_ON_CLOSE, NULL);
  DeviceIoControl(file, IOCTL_DISK_GET_DRIVE_GEOMETRY, NULL, 0, &s,
                  sizeof(s), &dummy, (LPOVERLAPPED) NULL);
  CloseHandle(file);
  block_size = (ULONG) s.BytesPerSector;
}
#endif
```

5.6. copy_files: Copia todos os arquivos de origem para destino

De posse da função que copia um só arquivo, precisamos definir uma função para copiar todos os arquivos de dentro d eum diretório para outro recursivamente. Isso é algo trabalhoso, pois precisamos listar todo o conteúdo de um diretório para obter seus arquivos. Como fazer isso varia dependendo do sistema operacional.

Em sistemas Unix a função usará `readdir` para ler o conteúdo de arquivos:

Seção: Funções auxiliares Weaver (continuação):

```
#if !defined(_WIN32)
int copy_files(char *orig, char *dst){
  DIR *d = NULL;
  struct dirent *dir;
  d = opendir(orig);
  if(d){
    while((dir = readdir(d)) != NULL){ // Loop para ler cada arquivo
      char *file;
      file = concatenate(orig, "/", dir -> d_name, "");
      if(file == NULL){
        return 0;
      }
#if (defined(__linux__) || defined(_BSD_SOURCE)) && defined(DT_DIR)
      // Se suportamos DT_DIR, não precisamos chamar a função 'stat':
      if(dir -> d_type == DT_DIR){
#else
      struct stat s;
      int err;
      err = stat(file, &s);
      if(err == -1) return 0;
      if(S_ISDIR(s.st_mode)){
#endif
      // Se concluirmos estar lidando com subdiretório via 'stat' ou 'DT_DIR':
        char *new_dst;
        new_dst = concatenate(dst, "/", dir -> d_name, "");
        if(new_dst == NULL){
          return 0;
        }
        if(strcmp(dir -> d_name, ".") && strcmp(dir -> d_name, "..")){
          if(directory_exist(new_dst) == NAO_EXISTE) mkdir(new_dst, 0755);
          copy_files(file, new_dst);
        }
        free(new_dst);
      }
      else{
        // Se concluimos estar diante de um arquivo usual:
```

```
        copy_single_file(file, dst);
      }
    free(file);
    } // Fim do loop para ler cada arquivo
    closedir(d);
  }
  return 1;
}
#endif
```

E isso requer inserir o cabeçalho:

Seção: Cabeçalhos Incluídos no Programa Weaver:

```
#if !defined(_WIN32)
#include <dirent.h> // readdir, opendir, closedir
#endif
```

No Windows, não é necessário inserir nenhum cabeçalho novo que já não está inserido. A definição da função fica assim:

Seção: Funções auxiliares Weaver (continuação):

```
#if defined(_WIN32)
int copy_files(char *orig, char *dst){
  WIN32_FIND_DATA file;
  HANDLE dir = NULL;
  dir = FindFirstFile(orig, &file));
  if(dir !== INVALID_HANDLE_VALUE){
    // The first file shall be '.' and should be safely ignored
    do{
      if(strcmp(file.cFileName, ".") && strcmp(file.cFileName, "..")){
        char *path;
        path = concatenate(orig, "\\", file.cFileName, "");
        if(path == NULL){
          return 0;
        }
        if(file.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY){
          char *dst_path;
          dst_path = concatenate(dst, "\\", file.cFileName, "");
          if(directory_exist(dst_path) == NAO_EXISTE)
            CreateDirectoryW(dst_path, NULL);
          copy_files(path, dst_path);
          free(dst_path);
        }
        else{ // file
          copy_single_file(path, dst);
        }
        free(path);
      }
    }while(FindNextFile(dir, &file));
  }
  FindClose(dir);
  return 1;
}
#endif
```