O *Framework* Weaver

Thiago “Harry” Leucz Astrizi

1 de Dezembro de 2009

Introdução

No ano de 2009, eu comecei a me interessar mais pelo *hobbie* de programar jogos em meu computador. Comecei criando algo bem simples no qual uma joaninha robótica tinha que se desviar de projéteis que descreviam movimentos parabólicos arremessados por caranguejos-robôs. Eu usei SDL para fazer o jogo.

Entretanto, fazer aquele projeto não me satisfez. Eu estava interessado em saber como as coisas realmente funcionavam — não em usar uma biblioteca já pronta. O SDL era uma biblioteca interessante. Mas não tinha muito da minha personalidade nela. Outra coisa que me incomodava eram as dependências que ela trazia. Isso me levou a tentar construir a minha própria biblioteca com funções que permitissem a criação de programas orientados à eventos (como jogos).

A primeira versão da Weaver rodou em meu finado laptop. Ela não rodava por cima do X, mas por cima do *framebuffer* do *kernel*. Criar funções para desenhar pontos e linhas foi trabalhoso. Entretanto, logo percebi que eu estava rumando para um destino não muito útil. O *framebuffer* era lento demais e seria frustrante trabalhar somente com uma fração tão pequena do potencial da minha máquina. Assim, a primeira Weaver foi destruída e comecei a escrever uma segunda versão que deveria rodar por cima do servidor X.

A segunda versão da minha biblioteca passou muito tempo em encubação. Foram muitas semanas de escritas de código, reescritas, acidentes que me fizeram perder tudo e mais reescrita. Nunca antes em um projeto eu fui tão exigente e paguei tão caro por isso. Por duas vezes, tive que jogar fora milhares de linhas de código por causa do projeto não estar do jeito que eu desejava. O projeto ainda não estava com a minha personalidade, o meu jeito.

Por fim, por volta de junho, uma versão suficientemente estável que me permitiu criar uma versão de *Pong* ficou pronta. No final de julho eu já havia criado um clone de *Spacewar!*. E no final de novembro já havia um clone simplificado de *Desktop Tower Defense* usando sprites e animações.

Weaver é uma biblioteca em constante desenvolvimento. Cada vez que eu inicio um projeto novo com ela, ela fica maior e melhor. Ela se alimenta de meu trabalho e se desenvolve desta forma. Ao perceber que uma importante

função não está disponível, eu a alimento com mais linhas de código.

Ela não é uma biblioteca de criação de jogos como todas as outras. Ela tem as seguintes características que a tornam especial:

- É uma biblioteca estática. Raramente você irá rodar mais de um jogo ao mesmo tempo. Logo, bibliotecas compartilhadas não oferecem economia de memória como em programas convencionais.
- Por causa disso, seus produtos, depois de compilados rodam em outras máquinas com o mesmo Sistema Operacional sem a necessidade de dependências externas.
- É software livre. Você pode alterá-la e modificá-la à vontade.
- Tudo o que ela gera também é necessariamente software livre. Graças ao fato dela ser GPL e ser inserida estaticamente no código dos jogos.
- Ela é (ou tenta ser) muito bem-documentada. Desta forma, aprender a usá-la torna-se fácil e você ainda aprenderá como ela é feita caso queira alterá-la.
- Além de ser uma biblioteca, ela também é um programa e um arcabouço (*framework*). Basta digitar **weaver meujogo**, que um diretório chamado "meujogo" será criado com um código mínimo nescessário para criar um jogo.
- Ela é exclusiva para sistemas GNU/Linux.
- Possui vários recursos que agilizam a criação de novos projetos e a criação de extensões em projetos já existentes. Tudo isso na lingüagem C.

Nos próximos capítulos deste documento, você encontrará um guia completo de como instalar e usar a Weaver. No apêndice existe também um guia de referência para todas as funções presentes na API Weaver. Divirta-se!

Capítulo 1

Instalando Weaver em seu Computador

Weaver foi projetada para funcionar em sistemas GNU/Linux. Se você não utiliza este sistema, talvez você até consiga instalar o programa, mas pode ser um tanto difícil. Você terá que tentar por sua própria conta e risco. Se você usa GNU/Linux, então o processo será bastante simples.

A primeira coisa que você deve obter é um arquivo compactado (possivelmente em tar.gz ou tar.bz2) com o código-fonte da Weaver. Se você está lendo isso, possivelmente você já obteve o código-fonte. Caso contrário, tente procurar utilizando algum mecanismo de busca na Internet.

Descompacte o arquivo, vá para dentro do diretório obtido e use o comando:

```
make install
```

Caso algum problema apareça nesta parte, provavelmente estão faltando algumas dependências. Provavelmente os arquivos de desenvolvimento para Xlib, Vorbis ou ALSA. Instale os pacotes que estiverem faltando antes de continuar.

Feito isso, o *framework* será instalado na sua máquina. O script Weaver será colocado por padrão em /usr/bin. O código da API e todos os arquivos necessários para gerar projetos iniciais serão colocados em /usr/share/weaver.

Vamos testar e ver se tudo está funcionando? Basta digitar no terminal:

```
weaver dummy
```

Depois deste comando, será criado um novo diretório chamado "dummy" que contém um código mínimo necessário para se criar um projeto Weaver. Entre neste diretório e use o comando:

```
make
```

O projeto será compilado. Se você executá-lo, encontrará um jogo vazio que será encerrado assim que qualquer tecla de seu teclado for pressionada. Se você conferir o código deste projeto no arquivo "src/game.c", encontrará um código como este:

```
#include "weaver/weaver.h"
#include "game.h"
```

Isso chama os cabeçalhos necessários para o seu jogo. O "weaver/weaver.h" é o mais importante. Ele declara todas as funções presentes na API Weaver. O "game.h" está vazio por enqüanto.

```
int main(int argc, char **argv){
  awake_the_weaver(); // Initializing Weaver API
```

Esta é a função principal e abaixo dela, uma função que inicializa a API Weaver. Todos os projetos feitos usando o *framework* Weaver devem começar com ela.

```
  // Main loop
  for(;;){
    get_input();
    if(keyboard[ANY]){
      break;
    }

    weaver_rest(10000000);
  }
```

Este é o loop principal. Um loop infinito dentro do qual o seu jogo irá rodar. No começo de cada iteração do loop, deve-se verificar se o usuário pressionou alguma tecla, mecheu o mouse, clicou em algo, etc. Isso é feito com a função "get_input()".

Em seguida, devemos reagir de acordo caso o usuário tenha apertado alguma tecla relevante. Neste caso, verificamos se qualquer (ANY) tecla foi pressionada e saímos do loop principal se isso aconteceu.

Caso contrário, continuamos dentro do loop. Invocamos a função "weaver_rest()" que faz duas coisas: primeiro, ela fica sem fazer nada pelo número de nanossegundos passado como argumento. Neste caso, 10000000 de nanossegundos é equivalente a 0,1 segundos. Isso faz com que o jogo não gaste processamento desnecessariamente por causa do seu loop infinito. Neste tempo, o seu computador tem um tempo livre para cuidar de outros processos que podem estar sendo executados.

Outra coisa que esta função faz é calcular a quantos frames por segundo o seu jogo está rodando. Ele armazena o resultado na variável global "fps".

Neste caso, o valor deve ser de 99 frames por segundo, já que o jogo está vazio e nada está sendo feito.

```
  may_the_weaver_sleep();
  return 0;
}
```

Finalmente, este é o fim do código do programa. Ele encerra a API Weaver e encerra o programa retornando o código 0 (de sucesso) para o seu Sistema Operacional.

Capítulo 2

Fazendo uma Bolinha Quicar na Tela

Vamos então começar a programar. Comecemos com o equivalente ao "Hello World" do mundo dos jogos. Vamos fazer uma bolinha ficar quicando pela sua tela. Ela ficará se locomovendo infinitamente até você encerrar o programa pressionando ESC.

Primeiro, comece um projeto digitando:

```
weaver ball
```

Abra o arquivo ball/src/game.c e adicione o cabeçalho:

```
#define RADIUS 30
```

Este é o raio da bolinha que ficará se movendo pela tela. No exemplo, estamos deixando ele com 30 pixels. Você pode deixá-la maior ou menor se preferir.

Para este "jogo", vamos precisar armazenar as coordenadas do centro da bolinha, bem como o quanto ela irá se deslocar horizontal e verticalmente a cada *frame*. Para isso, comece declarando no começo da função principal as variáveis necessárias:

```
int ball_x, ball_y, dx, dy;
```

Depois da função "awake_the_weaver()", nós já teremos obtido algumas informações que nos serão úteis. Como por exemplo, o tamanho da nossa tela. Só então poderemos inicializar os valores que declaramos:

```
ball_x = window_width / 2;
ball_y = window_height / 2;
dx = dy = 2;
```

Isso fará com que a nossa bolinha comece no centro da tela e se desloque a cada *frame* 2 pixels horizontalmente e 2 pixels verticalmente. As variáveis "window_width" e "window_height" são globais e são inicializadas pelo "awake_the_weaver()".

Agora, dentro do *loop* principal, depois do código que lida com o encerramento do programa caso o usuário pressione alguma tecla, adicione uma função para apagar a bolinha da tela pintando ela de preto:

```
draw_circle(circle_x, circle_y, RADIUS, BLACK);
```

Em seguida, coloque o código que controla o deslocamento da bolinha:

```
ball_x += dx;
ball_y += dy;
```

Agora vamos colocar código que controla o que acontece quando a bolinha atinge os limites da tela:

```
if(ball_y < RADIUS || ball_y > window_height - RADIUS)
   dy *= -1;
if(ball_x < RADIUS || ball_x > window_width - RADIUS)
   dx *= -1;
```

E finalmente, depois de apagarmos a bolinha e mudarmos sua posição, é hora de desenhar ela de novo:

```
draw_circle(circle_x, circle_y, RADIUS, RED);
```

As macros BLACK e RED vistas acima são definidas pela API Weaver. Além delas, existem muitas outras (GREEN, YELLOW, ORANGE, PINK, CYAN, ...). Para ver uma lista, basta ver o arquivo src/weaver/display.h.

Bem, o seu jogo está pronto. Basta agora compilá-lo com um `make` e executar o programa `ball` gerado. Você também pode instalá-lo com um `make install` se quiser.

Como pode ver, fazer um programa com uma bolinha se deslocando é muito fácil com a Weaver. Foram só 12 linhas de código. Para uma versão mais elaborada que tenha som e uma bolinha com textura, consulte o diretório `examples` que vem junto com o código-fonte da Weaver.