# Guia Básico do Nanvix: Instalação, Compilação, Execução e Depuração

Pedro H. Penna, Fernando Jorge Mota e Márcio Castro
Universidade Federal de Santa Catarina

## 1 Introdução

Antes de começar a *hackear* o Nanvix, é fundamental que você se familiarize com o modo como os arquivos do projeto estão organizados, com quais ferramentas de desenvolvimento você vai lidar e com o processo de compilação. Nesse primeiro projeto, você vai trabalhar em todas essas tarefas e, ao final, estará pronto para começar o trabalho duro.

É importante notar os scripts fornecidos para instalação e execução do Nanvix foram testados em distribuições Ubuntu e Lubuntu. Portanto, recomendamos fortemente que você utilize uma dessas duas distribuições. Certamente o Nanvix poderá ser compilado e executado em outras distribuições. Porém, será preciso em alguns casos realizar modificações nos scripts de instalação.

Inicialmente mostraremos como baixar o código fonte do Nanvix e discutiremos sua estrutura de diretórios (Seção 2). Em seguida, mostraremos como instalar as ferramentas de desenvolvimento que permitirão a sua compilação (Seção 3). Posteriormente, mostraremos como compilar e executar o Nanvix (Seção 4). Então, apresentaremos uma maneira interessante de depurar o Nanvix com uso do GDB (Seção 5). Por fim, mostraremos duas alternativas de utilização do Nanvix através do uso de Máquinas Virtuais (VMs): em uma máquina local (Seção 6) e na nuvem (Seção 7).

## 2 Código Fonte e Estrutura do Projeto

O primeiro passo para utilizar o Nanvix é baixar o seu código fonte. Para isso, basta clonar o seu repositório de desenvolvimento da seguinte forma[1]:

```
git clone https://github.com/nanvix/nanvix
```

Dentro do diretório do Nanvix você encontrará uma série de arquivos e diretórios. Por tratar-se de um projeto ligeiramente grande e complexo, o Nanvix é organizado em uma hierarquia de diretórios, que é detalhada a seguir:

- `bin` conterá o binário do *kernel* e utilitários, depois de terem sido compilados.
- `doc` contém toda a documentação do Nanvix, que inclui manuais do sistema, de bibliotecas e utilitários; orientações gerais para desenvolvimento; e documentação de APIs.
- `doxygen` contém arquivos de configuração da ferramenta Doxygen, que gera documentação das APIs do Nanvix diretamente do código fonte.
- `include` contém os arquivos-cabeçalhos de escopo global, tanto de sistema quanto de biblioteca.
- `lib` conterá todas as bibliotecas estáticas e dinâmicas, depois de terem sido compiladas.

[1] Caso você não tenha o `git` instalado, você poderá instalá-lo da seguinte forma: `sudo apt-get install git`

- `src` contém o código fonte do Nanvix.
- `tools` contém todas as ferramentas e *scripts* necessários para compilar o Nanvix.

## 3 Instalação das Ferramentas e Ambiente de Desenvolvimento

As ferramentas utilizadas no desenvolvimento do Nanvix, como no desenvolvimento de qualquer outro sistema operacional, se diferem das utilizadas no desenvolvimento da maioria dos outros tipos de software. Em primeiro lugar, as ferramentas de compilação devem ser compatíveis com a plataforma alvo. Por exemplo, o Nanvix foi projetado para a plataforma x86, portanto as ferramentas utilizadas para compilar o sistema devem ser capazes de gerar código de máquina para essa plataforma. Em segundo lugar, quando trabalha-se em nível de *kernel*, o ambiente de desenvolvimento não provê quaisquer tipo de bibliotecas padrões. Em terceiro lugar, para testar o sistema deve-se utilizar uma máquina dedicada, seja ela real ou virtual. Finalmente, em quarto lugar, as ferramentas de *debugging* disponíveis são restritas.

Para o desenvolvimento do Nanvix, você utilizará duas ferramentas: a *toolchain* GCC-x86, uma coletânea de utilitários que inclui compilador, *assembler* e *linker*; e o *Bochs*, um emulador para plataforma x86. Para instalar as ferramentas de forma automática execute os seguintes comandos **a partir do diretório raiz do projeto**:

```
sudo apt-get update
sudo apt-get install make
sudo bash tools/dev/setup-toolchain.sh
sudo bash tools/dev/setup-bochs.sh
sudo reboot now
```

É esperado que o processo de instalação das ferramentas demore um certo tempo (vários minutos).

## 4 Compilação e Execução

Uma vez que as ferramentas de desenvolvimento necessárias para compilar o Nanvix foram devidamente instaladas, compilar o sistema torna-se uma tarefa simples. Do diretório raiz do projeto, invoque os seguintes comandos **a partir do diretório raiz do projeto**:

```
make nanvix
make image
```

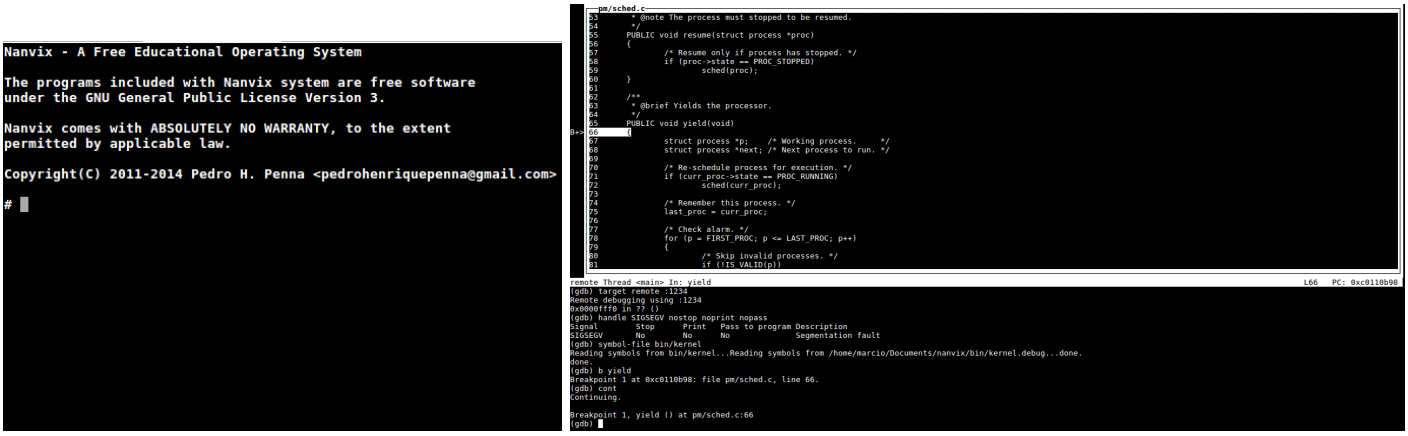
O primeiro comando realiza a compilação do Nanvix. Terminado esse processo, todos os binários devem ter sido criados com sucesso. O segundo comando gera uma imagem[2] do sistema, criando-se assim o arquivo `nanvix.img` no diretório raiz. Esse arquivo é a imagem do sistema e será usado no processo de *boot*, a seguir.

Finalmente, inicie o Nanvix no *Bochs*, dando *boot* pela imagem de sistema gerada. Para fazer isso de forma automática, execute o seguinte comando **a partir do diretório raiz do projeto**:

```
bash tools/run/run.sh
```

Pronto! Em caso de sucesso, você verá o Nanvix sendo inicializado. Após a inicialização, a tela principal do terminal do Nanvix está pronta para utilização, conforme mostra a Figura 1(a).

[2]Se você usa a distribuição *Arch Linux* talvez seja necessário instalar o pacote `cdrtools` para criar a imagem. Para isso, execute o seguinte comando: `pacman -S cdrtools`




(a) Nanvix após inicialização.

(b) GDB após um *breakpoint* na função `yield()` do *kernel*.

Figura 1: Terminais do Nanvix e do GDB.

**IMPORTANTE!**

O *Bochs* será executado diretamente na tela do terminal. Para finalizar a execução do emulador *Bochs* corretamente, pressione primeiramente CTRL+Z para pausar a execução do processo *Bochs*. Então, digite o seguinte comando no terminal para finalizar a execução do *Bochs*: `kill -HUP %1`.

# 5 Depurando o Nanvix

Para depurar Nanvix, o que pode ser muito útil na resolução de *bugs* e outros problemas que surgirão durante o desenvolvimento de código no *kernel*, será necessário o uso do GDB, cujo uso e integração serão definidos nessa seção. Caso você esteja numa conexão remota (e.g., usando `ssh`), é extremamente sugerido o uso do `tmux` para evitar a necessidade de criar mais de uma conexão `ssh`, visto que quando o *Bochs* está rodando ele assume o comando do terminal inteiro. Nesse caso, instale o `tmux` usando o comando:

```
sudo apt-get install tmux
```

Após a instalação, execute o comando `tmux` e pressione CTRL+B+% para abrir dois *panes*. Em seguida, use CTRL+B+(seta para direita ou seta para a esquerda) para alterar livremente entre os terminais. Mais informações podem ser vistas diretamente no manual do `tmux` disponível em: `https://leanpub.com/the-tao-of-tmux/read`.

Para rodar o Nanvix em modo depuração (*debug*) e ativar o suporte ao GDB utilize o seguinte comando em um terminal:

```
bash tools/run/run.sh --debug
```

Depois, inicialize o GDB **em um outro terminal** a partir do diretório raiz do projeto, usando o seguinte comando:

```
gdb --tui
```

O argumento `--tui` é opcional mas muito interessante. Com ele você habilitará o suporte a uma interface de usuário de modo texto para visualizar o código fonte durante a depuração, como mostrado na Figura 1(b).

Para começar a depurar, execute os seguintes comandos no terminal do GDB:

```
target remote :1234
handle SIGSEGV nostop noprint nopass
cont
```

Segue uma explicação breve aos comandos informados acima:

1. Conecta-se ao *Bochs*, que agora mostrará que há um cliente conectado;
2. Define que *page faults* e outras interrupções frequentes devem ser ignoradas completamente pelo GDB. O `SIGSEGV` aqui é apenas um *alias* para representar essas outras interrupções que são normais durante a execução da máquina;
3. Define que a máquina deve continuar sua execução.

Pronto! Em caso de sucesso você verá o GDB conectado ao *Bochs*, como mostra a Figura 1(b). Após a execução do comando `cont`, o terminal do Nanvix ficará disponível na tela do *Bochs* e o **GDB ficará bloqueado**. Para liberar o terminal do GDB novamente, aperte **CTRL+C** no terminal do GDB. Dessa forma, o GDB pausará a execução do Nanvix novamente e permitirá o controle da execução em modo *debug* através do GDB.

Durante a compilação do Nanvix, diversos arquivos contendo as tabelas de símbolos de funções do *kernel* e de programas de sistema e de usuário são criados. Porém, essas tabelas de símbolos não são carregadas automaticamente. Para carregar a tabela de símbolos contendo informações sobre todas as funções do *kernel*, você pode usar o comando:

```
symbol-file bin/kernel
```

Alternativamente, se o seu objetivo for depurar algum programa específico dentro do Nanvix será necessário carregar a tabela de símbolos do programa desejado. Por exemplo, para carregar a tabela de símbolos do programa `ls`, informe o caminho para o executável dentro da pasta `bin`:

```
symbol-file bin/ubin/ls
```

**IMPORTANTE!**

Não é recomendável usar mais de uma tabela de símbolos ao mesmo tempo, pois nesse caso o GDB não será capaz de diferenciar corretamente os símbolos. Logo, ao realizar esse comando em um terminal onde a tabela de símbolos do *kernel* já foi carregada o GDB perguntará se a tabela de símbolos deve ser substituída, o que é necessário para depurar um programa do Nanvix.

Para definir *breakpoints*, você pode usar o comando `b` **depois da tabela de símbolos ser carregada**. Para depurar a função `yield`, por exemplo, é possível simplesmente fazer:

```
b yield
```

No lugar do nome da função, você pode também definir uma linha onde o GDB deve definir o *breakpoint*. Dessa forma, você pode usar o seguinte comando para definir um *breakpoint* na linha 143 do arquivo **main.c**, por exemplo:

```
b main.c:143
```

Observe que a função de *breakpoint* só funciona bem se você carregar a tabela de símbolos usando o comando `symbol-file`, anteriormente especificado, pois é a partir da tabela de símbolos que o GDB consegue identificar os arquivos e funções envolvidos.

Após especificar um *breakpoint*, você poderá utilizar o comando `cont` para informar o GDB que o mesmo deve parar ao acontecer a próxima ocorrência de qualquer *breakpoint* anteriormente definido. Então, a partir de um determinado *breakpoint*, é possível usar o comando `step` para saltar para a próxima instrução:

```
step
```

Você também pode saltar um número definido de instruções com uso de um parâmetro opcional do `step`. Por exemplo, utilize o seguinte comando para saltar 10 instruções de uma única vez:

```
step 10
```

O comando `step` permite realizar uma depuração do tipo *step-by-step*. Isso significa que quando uma função é invocada durante a depuração, o GDB fará o desvio para a primeira instrução dentro da função. Caso você queira saltar diretamente para o retorno da função antes de invocá-la, você poderá utilizar o comando `next`:

```
next
```

Além disso, durante a depuração, é interessante usar o comando `print` para imprimir variáveis que possam ser úteis para entender a execução do código. Dessa forma, para imprimir a variável global `curr_proc`, que está presente no *kernel* do Nanvix, é possível usar:

```
print curr_proc
```

Da mesma forma, é possível usar parte da sintaxe do C para explorar os diferentes atributos que a variável possa ter. Por exemplo, para imprimir o atributo `pid` de `curr_proc`, faça:

```
print curr_proc->pid
```

O comando acima imprime o PID do processo atual em execução (no caso do Nanvix, o PID é um atributo de `curr_proc`). Observe que o `print` também funciona em *breakpoints*, com variáveis locais, sendo portanto bastante útil para entender o que está acontecendo no código.

Por fim, para saber a sequência de chamadas atual, você pode usar o comando `bt`, que imprime o *back-trace* atual do código:

```
bt
```

Para mais informações a respeito do GDB, consulte o manual em: `https://sourceware.org/gdb/current/onlinedocs/gdb/`

**IMPORTANTE!**

O *Bochs* não suporta determinadas operações que o GDB suporta, como `watch` (para acompanhar o valor de uma variável ao longo do tempo), ou chamadas de função no `print`. Fazer isso resulta em erro, em uma instabilidade do sistema ou ambos. Portanto, sua utilização não é recomendada.

# 6 Nanvix em uma Máquina Virtual (VM)

Caso você não tenha uma máquina com Linux e não tenha interesse em instalá-lo diretamente na sua máquina, você poderá fazer uma instalação do Linux em uma Máquina Virtual (VM) rodando em um outro sistema operacional (e.g., Windows). Nesse caso, aconselhamos a utilização do *Virtual Box* (disponível em `www.virtualbox.org`). Após instalar o Virtual Box, aconselhamos a instalação do Lubuntu 16.04 (disponível em `http://lubuntu.net`) na VM, pois trata-se de uma distribuição relativamente leve e testada. Por fim, basta seguir os passos de instalação e compilação (Seções 2 a 4) do Nanvix no Lubuntu.