Guide d'utilisation de Nanvix: Installation, Compilation, Mise en route et Débogage

Pedro H. Penna, Fernando Jorge Mota, Márcio Castro
Universidade Federal de Santa Catarina (UFSC)

Jean-François Méhaut
Université Grenoble Alpes (UGA)

1 Introduction

Avant de commencer les développements, il est essentiel de vous familiariser avec l'organisation des fichiers du système Nanvix, les outils de développement que vous allez utiliser et le processus de génération du système. Pour cette première séance, vous travaillerez sur toutes ces tâches et, à la fin, vous serez prêt à démarrer le travail de développement.

Il est important de noter que les *scripts* fournis pour l'installation et l'exécution de Nanvix ont été testés sur les distributions *Ubuntu* et Lubuntu. Par conséquent, nous vous recommandons fortement d'utiliser une de ces deux distributions Linux. Nanvix peut certainement être compilé et exécuté avec d'autres distributions Linux. Cependant, dans certains cas, vous devrez apporter des modifications aux scripts d'installation pour les adapter à votre distribution.

Pour démarrer, nous allons vous montrer comment télécharger le code source de Nanvix et discuter de sa structure en répertoires (Section 2). Ensuite, nous allons vous montrer comment installer les outils de développement qui devront être compilés pour permettre la génération du système (Section 3). Ensuite, nous allons vous montrer comment compiler et mettre en route Nanvix (Section 4). Enfin, nous montrerons un moyen intéressant pour déboguer Nanvix en utilisant GDB (Section 5). Enfin, nous allons montrer une alternative d'utilisation de Nanvix avec l'utilisation de machines virtuelles (VMs) dans une machine locale (Section 6).

2 Code source et la hiérarchie Nanvix

La première étape pour utiliser Nanvix est de télécharger le code source. Pour ce faire, il suffit de cloner le dépôt de développement comme suit [1]

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

Dans le répertoire Nanvix, vous trouverez un certain nombre de fichiers et de répertoires. Parce que le système d'exploitation Nanvix couvre plusieurs aspects et qu'il est un peu complexe, Nanvix est organisé autour d'une hiérarchie de répertoires, qui est détaillée comme suit :

- `bin` contiendra le binaire du noyau *kernel* et les utilitaires, une fois qu'ils auront été compilés.
- `doc` contiendra la documentation de Nanvix avec des manuels sur le système, les bibliothèques, les utilitaires, les directives générales pour le développement ainsi que de la documentation des APIs.
- `doxygen` contient des fichiers de configuration pour l'outil Doxygen qui permettront la génération de la documentation des APIs Nanvix directement à partir du code source.

1. Si vous n'avez pas installé `git`, vous pouvez l'installer de la manière suivante : `sudo apt-get install git`

— `include` contient des fichiers .h de déclaration de constantes, de types et des prototypes des fonctions ou appels au système Nanvix.
— `lib` contiendra les biblithèques statiques ou dynamiques, une fois qu'elles auront été compilées.
— `src` contient le code source du noyau du système Nanvix ainsi que des utilitaires.
— `tools` contient les procédures et les *scripts* nécessaires à la compilation du système, des utilitaires ainsi que les bibliothèques statiques ou dynamiques.

3 Environnements et outils de développement

Les outils utilisés pour le développement du système Nanvix diffèrent de ceux utilisés pour le développement logiciels plus classiques.

D'abord, les outils de génération doivent être compatibles avec la plate-forme cible et son processeur. Par exemple, Nanvix a été conçu pour la plate-forme de processeur Intel x86 (IA32 pour architectures 32 bits). La conséquence est que les outils utilisés pour compiler le système Nanvix doivent être capables de générer du code machine pour cette plate-forme. Ensuite, lorsque vous travaillez au niveau du noyau (*kernel*), l'environnement de développement ne fournit aucun type de bibliothèque standard. Enfin, pour tester le système Nanvix, il faudra utiliser une machine dédiée, qu'elle soit réelle ou virtuelle. Enfin, les outils pour la mise au point *débogage* disponibles sont d'assez bas-niveau, mais extrêmement puissants.

Pour le développement de nouvelles fonctionnalités pour Nanvix, vous utiliserez deux outils :
— La *toolchain* gcc-x86, une collection d'utilitaires qui inclut un compilateur (*gcc*), un assembleur (*as*, et un éditeur de liens (*linker*, *ld*) ;
— Bochs, un émulateur de plate-forme Intel x86 doté d'un puissant *debogeur* intégré.

Pour installer automatiquement ces outils, exécutez les commandes suivantes avec un mode privilégié d'exécution (`sudo`). Il y a deux scripts à exécuter qui sont dans le répertoire `tools/dev`.

```
sudo bash tools/dev/setup-toolchain.sh
sudo bash tools/dev/setup-bochs.sh
sudo reboot now
```

Le processus de génération et d'installation des outils peut prendre un certain temps (plusieurs minutes). Vous rebooterez (*reboot*) votre machine après l'installation des outils de développement.

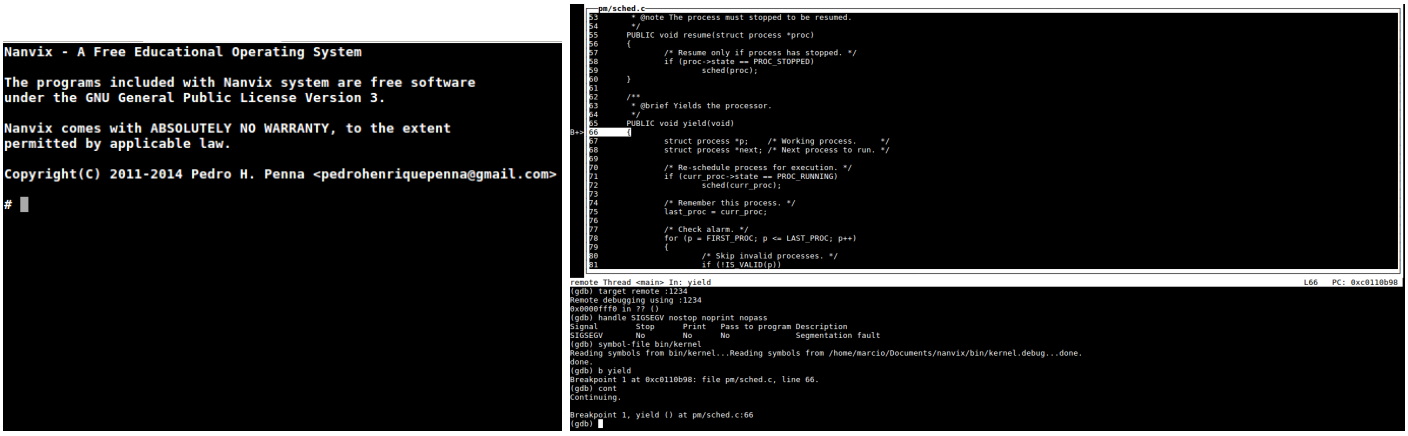
4 Compilation et mise en route de Nanvix

A partir du moment où les outils pour compiler le Nanvix auront été correctement installés, la compilation du système Nanvix devient une tâche simple. En vous positionnant dans le répertoire racine du projet, vous appelerez l'utilitaire `make` avec comme argument `nanvix`. Une fois cette commande exécutée, tous les fichiers binaires auront été créés avec succès.

Pour générer une image du système Nanvix, vous allez une nouvelle fois appeler l'utilitaire `make` à partir du répertoire racine, mais cette fois avec l'argument `image`. Cela va entraîner la création du fichier `nanvix.img` dans le répertoire racine. Ce fichier représente l'image système et sera utilisé dans la phase de démarrage (*boot*) décrite dans le paragraphe suivant.

```
make nanvix
make image
```

Enfin, vous allez démarrer le système Nanvix dans Bochs en invoquant le *boot* sur l'image système générée. Pour ce faire automatiquement, exécutez, à partir du répertoire racine, le *script* `run.sh` situé dans le répertoire `tools/run`.



(a) Après le démarrage (boot) de Nanvix

(b) [GDB après un *breakpoint* dans la fonction `yield` du *kernel*.

FIGURE 1 – Fenêtres avec Nanvix et GDB.

La première commande compile le Nanvix. Une fois cette commande terminé, tous les fichiers binaires doivent avoir été créés avec succès. La seconde commande génère une image [2] à partir du système, créant ainsi le fichier texttt nanvix.img dans le répertoire racine. Ce fichier est l'image système et sera utilisé dans le processus *boot* ci-dessous.

Enfin, lancez le fichier dans *Bochs*, en donnant *boot* sur l'image système générée. Pour ce faire automatiquement, exécutez la commande **suivante à partir du répertoire racine du projet** :

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

En cas de succès, vous verrez Nanvix en cours d'initialisation (boot). Après l'initialisation, l'écran principal du terminal Nanvix est prêt à l'emploi, comme illustré sur la figure 1(a).

IMPORTANT!

Bochs sera exécuté directement dans le terminal. Pour terminer correctement l'exécution de l'émulateur *Bochs*, appuyez d'abord sur CTRL Z pour mettre en pause l'exécution du processus *Bochs*. Ensuite, entrez la commande suivante sur le terminal pour terminer l'exécution de *Bochs* : `kill -HUP %1`.

5 Débogage de Nanvix

Pour déboguer Nanvix, qui peut être très utile pour identifier et corriger des *bogues* et autres problèmes qui se posent dans le code en cours de développement *noyau*, l'utilisation de GDB est nécessaire, l'utilisation et l'intégration seront présentées dans la section actuelle.

Pour exécuter en mode de débogage Nanvix et activer le support GDB utiliser la commande suivante dans un terminal :

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

Ensuite, démarrez GDB **sur un autre terminal** à partir du répertoire racine du projet, en utilisant la commande suivante :

2. Si vous utilisez la distribution *Arch Linux*, vous devrez peut-être installer le paquetage `cdrtools` pour créer l'image. Pour ce faire, exécutez la commande suivante : `pacman -S cdrtools`

```
gdb --tui
```

Le paramètre `-tui` est optionnel mais très intéressant. Avec lui, vous activez la prise en charge d'une interface utilisateur en mode texte pour afficher le code source pendant le débogage, comme illustré dans la figure 1(b).

Pour commencer le débogage, exécutez les commandes suivantes sur le terminal GDB :

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

Voici une brève explication des commandes données ci-dessus :

1. Il se connecte au *Bochs*, qui affichera qu'un client est connecté ;
2. Définit que les défauts de page et autres interruptions fréquentes doivent être complètement ignorées par GDB. SIGSEGV est juste un alias pour représenter les autres interruptions qui sont normales pendant l'exécution de la machine ;
3. Définit la machine pour continuer à fonctionner.

En cas de succès, vous verrez comment GDB est connecté à Bochs, comme le montre la figure 1(b). Après l'exécution de la commande `cont`, le terminal Nanvix sera disponible sur l'écran Bochs et GDB sera bloqué. Pour relâcher le terminal GDB, appuyez sur **CTRL C** sur le terminal GDB. De cette façon, GDB va à nouveau interrompre l'exécution de Nanvix et permettre le contrôle de l'exécution en mode *debug* via GDB.

Lors de la compilation de Nanvix, plusieurs fichiers contenant les tables de symboles du *kernel* et des programmes système et utilisateur ont été créés. Cependant, ces tables de symboles ne sont pas chargées automatiquement. Pour charger la table de symboles contenant des informations sur toutes les fonctions de *kernel*, vous pouvez utiliser la commande :

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

D'autre part, si votre objectif est de déboguer un programme spécifique s'exécutant au dessus de Nanvix, il sera nécessaire de charger la table de symboles du programme désiré. Par exemple, pour charger la table de symboles du programme, tapez le chemin d'accès à l'exécutable dans le dossier `bin` :

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

IMPORTANT!

Il n'est pas recommandé d'utiliser plus d'une table de symboles à la fois, car GDB ne sera pas capable de différencier les symboles correctement. Ainsi, lorsque vous exécutez cette commande sur un terminal où la table de symboles de *kernel* a déjà été chargée, GDB demandera si la table de symboles doit être remplacée, ce qui est nécessaire pour déboguer un programme Nanvix.

Pour insérer des *breakpoints* (points d'arrêt), vous pouvez utiliser la commande `b` **après le chargement de la table de symboles**. Pour déboguer la fonction texttt yield, par exemple, vous pouvez simplement faire :

```
b yield
```

À la place du nom de la fonction, vous pouvez également définir une ligne où GDB doit définir *breakpoint*. De cette façon, vous pouvez utiliser la commande suivante pour définir un *breakpoint* sur la ligne 143 du fichier

main.c, de la manière suivante :

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

Notez que le *breakpoint* ne fonctionne correctement uniquement si vous chargez la table de symboles à l'aide de la commande `symbol-file` précédemment spécifiée, car c'est à partir de la table de symboles que GDB peut identifier les fichiers et fonctions concernées.

Après avoir positionné un *breakpoint*, vous pouvez utiliser la commande `cont` pour informer GDB qu'il doit s'arrêter lorsque la prochaine occurrence de *breakpoint* précédemment positionnée est rencontrée. Ensuite, à partir d'un point d'arrêt donné, vous pouvez utiliser la commande `step` pour passer à l'instruction suivante :

```
step
```

Vous pouvez également ignorer un certain nombre d'instructions en utilisant un paramètre optionnel de `step`. Par exemple, utilisez la commande suivante pour ignorer 10 instructions uniques :

```
step 10
```

La commande `step` vous permet d'effectuer un débogage de type *step-by-step*. Cela signifie que lorsqu'une fonction est appelée pendant le débogage, GDB ignore la première instruction de la fonction. Si vous voulez passer directement à la fonction return avant de l'appeler, vous pouvez utiliser la commande `next` :

```
next
```

En outre, pendant la phase de débogage, il est intéressant d'utiliser la commande `print` pour imprimer les variables qui pourraient être utiles pour comprendre l'exécution du code. Ainsi, pour imprimer la variable globale `curr_proc`, qui est présente dans le Nanvix *kernel*, vous pouvez utiliser :

```
print curr_proc
```

De même, vous pouvez utiliser une partie de la syntaxe C pour explorer les différents attributs (champs) que la variable peut avoir. Par exemple, pour imprimer le champ `pid` de `curr_proc`, faites :

```
print curr_proc->pid
```

La commande ci-dessus imprime le PID (*Process IDentifier*) du processus en cours d'exécution (dans le cas de Nanvix, le PID est un attribut de `curr_proc`). Notez que `print` fonctionne aussi sur *breakpoints*, avec des variables locales, il est donrc très utile de comprendre le comportement du code.

Enfin, pour visualiser la pile des appels de fonctions, vous pouvez utiliser la commande `bt`, qui affiche le *back-trace* actuel du code :

```
bt
```

Pour plus d'informations sur GDB, consultez le manuel à l'adresse : https ://sourceware.org/gdb/current/onlinedocs/g

IMPORTANT!

Bochs ne supporte pas certaines opérations GDB, comme par exemple `watch` (pour suivre la valeur d'une variable dans le temps), ou les appels de fonction dans la commande `print`. Cela entraîne une erreur, une instabilité du système ou les deux. Par conséquent, leurs utilisations ne sont pas recommandées.

6 Nanvix dans une machine virtuelle (VM)

Si vous n'avez pas de machine Linux et que vous ne voulez pas l'installer directement sur votre machine, vous pouvez installer Linux dans une machine virtuelle (VM) fonctionnant sur un autre système d'exploitation (par exemple, Windows). Dans ce cas, nous vous recommandons d'utiliser *Virtual Box* (disponible sur `www.virtualbox.org`). Après avoir installé Virtual Box, nous vous conseillons d'installer Lubuntu 16.04 (disponible sur http ://lubuntu.net) sur VM, car il s'agit d'une distribution relativement légère et testée. Enfin, il suffit de suivre les étapes d'installation et de compilation (Sections 2 et 4) de Nanvix dans Ubuntu.