

できる StarBED

—SpringOS を使って楽々大規模実験—

宮地利幸

Name : version - 0 - 1

1 はじめに

「StarBED で実験をしたいのだけど、SpringOS で何ができて、どうしたら使えるのかわからない」という話をよく聞きます。実は StarBED を SpringOS で駆動して実験を行うこと自体はそう難しくありません。本チュートリアルでは SpringOS の機能を簡単に確認して、SpringOS を利用するための各種手順をまとめます。それでははじめましょう。

2 SpringOS の基本的な機能

SpringOS は StarBED のノードを操作し、実験を行うために必要な設定やノードでのコマンド実行を行います。代表的な機能を以下に挙げます。

- ノードの電源投入、停止、再起動
- ノードの起動方法の変更
- ノードへの OS の導入
- スイッチ (VLAN) の設定
- ノードでのコマンド実行によるシナリオ実行
- 実験リソースの管理

StarBED では管理用のネットワークが用意されていて静的なネットワーク設定がなされています。このネットワークを使えばどんなときも (管理用ネットワークに関するトラブルや OS のハングアップなどが起きていなければ) 実験用ノードにアクセスできます。

2.1 ノードの電源投入、停止、再起動

SpringOS はノードの起動に、Wake on LAN (WoL) を用いています。WoL は同じセグメントに存在するノードに対してマジックパケットを送信して、対象ノードを起動する方法です。逆に言うと同じセグメントに接続されているノードしか起動できません。StarBED の管理用ネットワークは複数に分割されていますので、WoL のセグメントを別セグメントに中継する必要があります。これを行うのが WoL Agent です。WoL は電源の投入しか行えません。そこで実験用ノードで動作する SNMP エージェントを制御することで電源断や再

起動を行っています。このためにノードで動作するアプリケーションとして snmpmine を用意しています。

2.2 ノードの起動方法の変更

ノードの起動方法をいちいちそれぞれの BIOS の設定を変えたり、ブートローダの設定を変えていたら日が暮れてしまいます。これを、リモートから一括での変更を可能とするために、StarBED の PC ノードは必ず PXE によるネットワークブートを行います。このときに取得される bootloader を変更することで、起動する方法やパーティションを変更しています。1stbootloader は TFTP で転送され、tftpboot 以下の各ノード用の bootloader を変更するのが DMAN の役目です。各ノードが起動してきたときに IP アドレスの取得や、bootloader の取得を行うための情報は DHCP を用いて指定します。この bootloader は COIL として提供されています。

2.3 ノードへの OS の導入

ノードへの OS の導入は、ディスクイメージの導入またはネットワークブートによる起動方法を採用しています。

ディスクイメージによる方法は、まず一台のノードに必要な OS やアプリケーションをインストールし、このディスクの該当パーティションをバイナリイメージとして取り出し FTP サーバに保存、これを他のノードに分配することにより行います。多くの実験用ノードの構成は同じことが多いため、このような方法をとっています。また、それぞれのノードによって異なる設定は必要なコマンドをシナリオの一部として実行することによって行います。最初の一台にインストールした OS のイメージは pickup を用いて取り出します。ディスクイメージの書き込みのみ行いたい場合は wipeout で、実験の一部として OS 導入を行う場合は kuroyuri master(以下 master) によって書き込みが行われます。これら 3 つの利用者のインターフェースとなり SpringOS の様々な機能を利用して実験などを実行する要素群を、この後 ENCD(Experiment NodeConfiguration Driver) と呼びます。ディスクイメージの読み込み、書き込みを行う際には、対象のノードは専用のディスクレス OS を利用して起動し、この上で動作している NI が ENCD の指示にしたがって指定されたディスクイメージの操作を行います。NI が起動したときに、自分を管理するはずの ENCD がどこで動作しているかわからないので、施設で固定的に動作している FNCP をつかいます。ENCD は、自分が担当するノードの IP アドレスと自分の IP アドレスを FNCP に登録します。起動してきた NI は、FNCP に自分の IP アドレスに対応する ENCD の IP アドレスをといあわせることで ENCD に接続します。

ディスクレスで起動する場合は必要なイメージを用意してそこから起動するように設定します。ただし、PXE を利用したディスクレスシステムの実現方法は OS により設定が必要なファイルなどが様々で現在すべての OS に対応できているわけではありません。

2.4 スイッチ (VLAN) の設定

StarBED では物理的配線を変更しないため、VLAN および ATM を使って仮想的に L2 トポロジを生成します。master が SWMG にトポロジ変更依頼を出すことで、変更が行われます。現在の SpringOS は VLAN のみに対応しています。

2.5 シナリオ実行

master と kuroyuri slave(以下 slave) が協調することでシナリオを実行します。master は実験駆動単位管理用のノードで動作し、slave は実験用ノードで動作し、実験用のシナリオ(コマンドリスト)を実行します。ノードの協調は slave と master の間のメッセージ交換で実現しています。ある slave の動作を待って他の slave が何かしらのアクションをとるときも必ず master を経由したメッセージ交換が行われます。master でもシナリオが動いていて、このシナリオは基本的にはこのようなメッセージ交換の制御を行うためのものです。

またネットワーク設定のためには、各実験ノードで実験側 NIC への IP アドレスの設定が必須です。これはシナリオ実行の一部として行われます。ただし、各ネットワークインターフェースのデバイス名は OS によって異なります。場合によっては起動する毎に変わってしまうこともあるので、SpringOS では各インターフェースの MAC アドレスを元に IP アドレスを設定します。MAC アドレスとデバイス名のマッピングを取るために ifscan を利用します。

2.6 実験リソースの管理

実験を行うためには、ノードや VLAN 番号といった実験用のリソースを必要な分だけ実験に割り当てる必要があります。このために ERM がノードと VLAN 番号を管理していて、master の要求を満たすリソースを割り当てます。ERM は一つのリソースを複数の実験に割り当てないよう、その状態も保存しています。ノードの情報が必要な場合に master などは ERM に問い合わせを行います。

3 実験駆動単位管理用ノードの設定

説明に入る前に実験駆動単位という言葉だけ覚えておいてください。ある実験用のノードとその実験を管理するためのノードをあわせて実験駆動単位ということにします。そして、実験駆動単位管理ノードというのは、実験を管理するための kuroyuri master が動作するノードをさすことにします。このノードは実験が始まる時には用意されていることが必要で、ここで master を動作させることで実験が始まります。

さて、さっそく環境を作りましょう。ここまで見てきているいろいろな設定が必要に見えますが、利用者が行わなければいけないことは SpringOS のモジュール群を駆動して、実験自体を実行するモジュールである kuroyuri master(以下 master) のコンパイルと、実験用ノードのディスクイメージの用意、そして設定ファイルを記述することです。

実験駆動単位管理用のノードに kuroyuri master をコンパイルしましょう。実験駆動単位管理用のノードといっても、基本的に master が動作するだけです。各種 SpringOS の機能を CLI から制御できる sbpsh もインストールすると便利ですが、それだけですので、特に特別なノードや OS が必要というわけではありません。StarBED の実験用ノードを利用してもいいですし、手元のラップトップなどに master を入れても問題ありません。必要なことは master がコンパイルできる環境であることと、利用するノードが StarBED の管理用ネットワークに接続されていることです。OS は FreeBSD や MACOSX、LINUX などでコンパイルできることが確認できています。

3.1 SpringOS ソースの取得とコンパイル

さて、早速始めましょう。SpringOS のインストール先は `/usr/local/springos` とします。バイナリは `/usr/local/springos/bin` に置きますので、そこにパスを通しておくことにします。

SpringOS のソースコードを <http://www.starbed.org/> から拾ってきます。ここでは、download ページの snapshot の、`SpringOS-Release-Ix-rc6.tar.gz` を利用します。

さてダウンロードしてきた SpringOS を伸長して、コンパイルしましょう。ソースコードをおくところはどこでもいいですが、ここでは `/usr/local/springos/src` というディレクトリをつくってそこにおいています。

```
$ cd /usr/local/springos/src/
$ tar xzf SpringOS-Release-Ix-rc6.tar.gz
$ cd SpringOS-Release-Ix-rc6
$ ls
dck13-080411.tar.gz      fncp-071116.tar.gz      pqerm-080411.tar.gz
dman-071116.tar.gz      ifsetup-071116.tar.gz   swmg-080411.tar.gz
erm-071116.tar.gz       mine-071116.tar.gz
wolagent-071116.tar.gz
```

このうち `dck` を伸長してコンパイルします。

```
$ tar xzf dck13-080411.tar.gz
$ cd dck13-080411/src
$ sudo ./configure
$ sudo make
```

`master`, `sbpsh`, `pickup`, `wipeout` の各バイナリへ、`/usr/local/springos/bin` からシンボリックリンクを貼っておいてください。

つぎに `sbpsh` の設定ファイル `sbpsh.rc` を記述しておく必要があります。

```
set rm 127.0.0.1 1234
#set tftpdman 172.16.3.101 1236
#set wolagent 172.16.1.101 5959 172.16.1.0/23
#set wolagent 172.16.3.101 5959 172.16.3.0/23
#set wolagent 172.16.4.200 5959 172.16.4.0/24

set user starbed
set project starproj
```

図 1: `sbpsh.rc` のサンプル

図 1 に例を挙げますが、基本的に StarBED のスタッフに確認してください。IP アドレスなどが適切に設定されたものが用意されているはずです。ただし、最後の 2 行には予約時指定されたユーザとプロジェクトを記述してください。これで利用できるはずですが、起動方法は以下の通りですが、使い方は後で説明します。

```
$ sbpsh -r sbpsh.rc
```

そして ERM の設定、起動をします。本来はユーザが設定する必要が無いものなのですが、現状のシステムですと設定が必要になります。2008 年 6 月には ERM の設定が必要なくなるはずですが。

基本的に先ほどの tarball に含まれていた ERM の tarball を伸長しコンパイルします。そして、erm のソースディレクトリの下にある starbed-acl と starbed-project を書き換えます。

starbed-project の内容は以下のようになっています。

```
starbed:starpass:starproj
```

“:” で区切られた最初がユーザ名、真ん中がパスワード、そして最後がプロジェクト名です。ここをすきな物に変更してください。

次に starbed-acl を書き換えます。中身は以下のような物です。

```
permitrange node starbed sintcla001-208  
permitrange vlan starbed 800-810
```

一行目は利用できるノードの範囲です。予約したノードを記述してください。そして 2 行目は VLAN です。これも利用できる VLAN の範囲を確認して変更してください。3 カラム目は先ほど設定したユーザ名です。

さて、そうしたら ERM を起動しておきます。ログをたとえば /var/log/springos 以下に保存しておきましょう。

```
$ sudo -c 'sh ./erm -D ../data/starbed-resource \<\  
-U ../data/starbed-project \<\  
-A ../data/starbed-acl > /var/log/springos/erm.log 2>&1'
```

これだけで実験駆動単位管理用ノードの設定は終了です。次に実験用ノードの設定にうつります。

4 実験用ノードの設定

ここでは、実験用のノードには HDD に OS をインストールして利用することにします。現状の SpringOS は現状で UNIX 系 OS だけを対象にしています。SpringOS が動きそうな OS をつかってください。もしそれっぽいのに動かない！ということがあれば StarBED の開発チームにご連絡いただければ対応できるかもしれません。

このとき、インストール時には基本的に現在構築してあるパーティションテーブルをそのまま利用してください。そうでないと、対象のディスクイメージを他のノードに導入する際にもパーティションテーブルの変更が必要になってしまいます。現時点の SpringOS はパーティションテーブルの変更には対応していませんので、正常な起動ができない可能性があります。また、ブートローダーは HDD の最初ではなくて各パーティションの最初にインストールする必要があります。

4.1 SpringOS のコンパイル

SpringOS は実験駆動単位管理用ノードを設定した時にダウンロードしてきた tarball に含まれていたものを使います。実験駆動単位管理ノードのときとコンパイルの方法は一緒です。コンパイルが必要なパッケージ

は、dck(に含まれる slave), mine, ifsetup です。実験九度運単位管理用ノードを設定したときと同様に、各バイナリへ/usr/local/springos/bin からシンボリックリンクを貼っておくと便利かもしれません。

4.2 slave と snmpmine の設定

kuroyuri slave と snmpmine はそれぞれサーバからの要求を受けてそれぞれの動作をするクライアントプログラムです。kuroyuri slave(以下 slave) は master から指定されたコマンドを実行することで実験シナリオをすすめ、snmpmine は電源の管理を行うために SNMP エージェントです。これらは特に設定を行う必要はなく、単に起動すれば OK です。OS 起動時に起動するようしておきましょう。以下は FreeBSD などで行うことができる/etc/rc.local です。

```
#!/bin/sh

/usr/local/springos/bin/slave -t -d > /var/log/springos/slave.log 2>&1 &
/usr/local/springos/bin/snmpmine &
```

4.3 ネットワークの設定

次に、管理側のネットワークインターフェースの設定を行います。DHCP でアドレスがつくように設定しておいてください。たとえば、FreeBSD では、/etc/rc.conf に以下の記述をします。

```
ifconfig_wm0="DHCP"
```

これに加えて、必要なアプリケーションをインストールします。ここでは、後で例題として netperf を実行することにしますので、netperf をインストールしておきます。FreeBSD なら ports などから、Linux なら rpm や apt などを使って簡単にインストールできると思います。

さて、この環境を使って簡単な実験を動かしてみましょう。次章では SpringOS のシナリオファイルの記述方法を解説します。

4.4 ディスクイメージの作成

さて今インストールした内容をバイナリイメージとして保存します。すでに実験駆動単位管理用ノードが用意できていますので、このノードから制御します。先ほど pickup と wipeout をインストールしましたね。これを使って作業します。

pickup は、ノードとパーティション番号を指定してその中身をバイナリイメージとして、ファイルサーバに保存します。これを行うためにたくさんのオプションをわたさないといけないのですが、面倒なので(少なくとも StarBED 用に)決まり切ったオプションはファイルに書いておくと便利です。このファイルを pickup.opt とします。例を以下に挙げます。

```
-u starbed
-p starpass
-j starproj
-r localhost:1234
```

```
-k 10.211.55.5:1236
-f 10.211.55.5:1238
-s 10.211.55.5
-K FreeBSD/ni05.fs:recover_system/kernel-groupf
-P recover_system/pxeboot-nohang
-w 10.211.55.5:5959:10.211.55.0/24
```

これも sbpsh.rc と同様に StarBED のスタッフにお願いしてもらってください。ただし -u、-j、-p、そして -s だけは変更が必要です。-u、-j、-p は予約時に割り当てられたユーザとパスワード、プロジェクトです。そして -s は pickup を実行する実験駆動単位管理用ノードの管理側ネットワークに接続されているネットワークインターフェースの IP アドレスを指定してください。pickup 実行時に同じオプションを渡すこともできます。

pickup の実行は以下のように行います。

```
pickup -F pickup.opt -X ftp://install:install@10.211.55.5 sintcla209:1
```

-F で先ほど作った設定ファイルの指定、そして -X はディスクイメージの保存先です。ここでは ftp で保存したイメージをアップロードしていますね。そして、最後がターゲットのノードとパーティション番号です。例ではグループ A の 209 番目のノードの 1 番目のパーティションを保存します。(実際には A の 209 番目のノードは存在しません、このまま実行して事故が起きることを予防するためにこうしてあります) ディスクイメージは -X で指定されたノードに保存されます。ここでは install というユーザのホームディレクトリに保存されます。ファイル名は

ノード名-デバイス名-日付 + 時間.gz

デバイス名は、ディスクイメージを作成するための OS からみたデバイス名になりますので、実験ノードにインストールした OS からみたデバイス名と異なることが多いので注意してください。たとえば以下ようになります。

```
sintcla209-rad0s1-200805221550.gz
```

さて、これでディスクイメージなどの用意ができたので、シナリオファイルの記述を始めましょう。

ちなみに、書き込みのテストを行いたい場合や、書き込みだけを行いたい場合は wipeout を利用することで目的を達成できます。以下のようにすると先ほど保存したファイルが sintcla210 のパーティション 1 に書き込まれます。設定ファイルは pickup と同じ物を使えますのでそのままです。typo じゃないですよ。

```
/usr/local/springos/bin/wipeout -F /usr/local/springos/etc/pickup.opt\  
-X ftp://starbed:starpass@10.211.55.5/sintcla209-rad0s1-200805221550.gz sintcla210:1
```

5 シナリオ記述

シナリオは実験の設定ファイルだと思ってください。我々は実験用ノードで実行される実験手順をシナリオと呼んでいて、基本的には各ノードで実行されるコマンドリストの集合だと思っていただければいいと思います。SpringOS のシナリオファイルには実行されるコマンドリストだけでなく、ノードの設定情報などの記述も行います。

```

rmanager ipaddr "127.0.0.1" port "1234"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.0.0/24"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/24"
wolagent ipaddr "172.16.4.253" port "5904" ipaddrrange "172.16.4.0/24"
fncp ipaddr "172.16.3.101"
tftpdman ipaddr "172.16.3.101"

nidiskimage "FreeBSD/ni04b.fs"
nikernel "recover_system/kernel_recover"
pxeloder "recover_system/pxeboot-nohang"
setuptimeout total 86400 warm 5

```

図 2: pre.sc の例

5.1 まずは動かしてみよう！

まずは、実験駆動単位管理用ノードで一台の実験用ノードを制御して、touch コマンドでファイルを作ってみましょう。

5.1.1 シナリオファイルを用意する

まずは master/slave の動作を制御するための設定ファイルの作成が必要です。実験駆動単位管理用ノードで pre.sc というファイルを作成します。図 2 に一例を示します。実験の実行にはこの pre.sc とそれぞれの実験用のシナリオファイルを用意します。実は pre.sc の内容をそれぞれのシナリオファイル中に記述しても全く問題がありません。でも、このような内容は環境が変わらない限り変更する必要がないので、別ファイルにしておいて使い回してしまおうというのが狙いだったりします。ですのでこれも StarBED チームに問い合わせせて最新のものをもらってください。

では、次に touch コマンドを実行するためのシナリオファイルを用意しましょう。図 3 がその中身です。user、project は予約時に指定されたものを指定しましょう。ここで encd は master のことですから、master が動作している実験駆動単位管理用ノードの IP アドレスで埋めます。(これも pre.sc に書いてしまっても問題ないかもしれませんが) sparnodemain と sparnoderatio は予備ノードに関する記述です。今回は予備ノードを確保しない設定となっています。

次にノードの設定です。実験を行うときには同じ設定のノードを使う場合が非常に多いです。SpringOS では設定を一つ書き、その設定を使うノードを何台用意するかを指定するといったオブジェクト指向的な記述方法を採用しています。ここでは、cclass というクラスを定義しており、そのインスタンスの client を 1 台用意しています。

nodeclass で定義した cclass の設定内容は、method で起動方法、partition で利用するパーティション、ostype で OS の種類、diskimage でインストールするディスクイメージを指定しています。今回は HDD にディスクイメージを導入するので、method は HDD、partition には先ほど保存したディスクイメージと同じ 1、diskimage には先ほど保存したファイルの場所を指定します。

nodeclass 内の scenario が肝心の touch の実行部分です。callw でフォアグラウンドモードでコマンドの実行を行います。call を使うとバックグラウンド実行します。

最後の nodeclass の外で指定されている scenario から始まる部分は、グローバルシナリオと呼んでいる全

```

user "starbed" "info@starbed.org"
project "starproj"

encd ipaddr "10.211.55.10"

sparenodemain 0
sparenoderatio 100

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.5:sintcla209-rad0s1-200805221550.gz"
    scenario {
        callw "/usr/bin/touch" "/tmp/huga"
    }
}

nodeset client class clclass num 1

scenario {
    sleep 10
}

```

図 3: touch.sc

体の実験の管理をするためのシナリオです。これについては後々解説しますので、現状では sleep 10 として 10 秒待機するコマンドを実行します。ちなみにグローバルシナリオに対して、各ノードで実行されるシナリオは、ノードシナリオと呼びます。

5.1.2 実験を実行する

というわけで先ほどのシナリオファイルを利用して実験を動かしてみましょう。まずは、仮想機械の流儀にしたがって、実験駆動単位管理用ノード、実験用ノード 2 台の合計 3 台を起動した状態にしておいてください。実際には実験駆動単位管理用のノードと実験用ノード 1 台でいいのですが、後でどうせ起動するので一緒に起動してしまいましょう。

実験の実行は以下のようにします。

```

$ cd /home/starbed/bin/
$ master -p starpass ../conf/sc/pre.sc ../conf/sc/touch.sc

```

-p は erm のパスワードです。実は pre.sc に記述してもいいのですが、セキュリティ上の理由からコマンドラインで入力しています。この状態でもあまりセキュリティ上よいとは言えませんが...

StarBED では KVM 装置をつかって各ノードのコンソールを確認することができます。ディスクイメージのダウンロードなどについては様々な情報が出力されるので、それが出力されているかですまず確認を行いましょう。ノードがそもそも起動してこなかったり、すぐに再起動してしまうなど、問題がありましたら設定ファイルを確認してください。間違いがなさそうなら StarBED のスタッフに確認してください。

5.1.3 確認

さて、実験が実行されたなら、OS のインストールとシナリオの実行がなされているはずです。ちゃんと動作したか確認しましょう。

まず、OS は意図したもので起動していたでしょうか？そして、touch で作成した/tmp/huga というファイルが存在するかを ls など確認しましょう。

作成されていませんか？作成されていれば成功です。何か問題があればこれまでの手順に間違いがある可能性があります。問題が解決しなければ StarBED チームのだれかに相談してください。

5.2 メッセージ交換を試みる

SpringOS はメッセージの交換によってノード間の同期を取っています。ノード A でなにかが起きた後にノード B で処理を行いたいときは、ノード A の slave が master にメッセージを送って、それを受け取った master がノード B にメッセージを送ることでノード B の slave が処理を開始するタイミングを知ります。すでに述べたグローバルシナリオは基本的にこのメッセージ制御を行うために利用されます。

ここではメッセージ交換の仕方と、その前に変数の使い方について勉強しましょう！

5.2.1 シナリオを書く

先ほどの touch のシナリオのノードシナリオ部分を図 4 のように、そしてグローバルシナリオ部分を図 5 のように書き換えます。(シナリオ全体は付録を参照してください)ここでは新しいシナリオファイルを msg.sc として保存しますね。

```
nodeclass clclass {
    < >
    scenario {
        send "setupdone"
        recv val
        callw "/bin/echo" val > "/tmp/huga"
        send "done"
    }
}
```

図 4: メッセージ交換ノードシナリオ

ノードシナリオでは、setupdone という内容のメッセージを送って (send)、次に master から送られてきたメッセージを変数 val に保存するように待機します (recv)。そして、何らかのメッセージを受け取ったら echo を実行してメッセージの中身を/tmp/huga に書いています。そしてこれらの処理が終わったら master に done というメッセージを送っています。

グローバルシナリオでは、client[0] から setupdone というメッセージを受信するまで待機します。client[0]? これ为什么呢。ノードを設定で clclass のインスタンスを client という名前で一台用意しました。SpringOS ではノード定義の時に指定した名前 + [+ 番号 +] で各ノードを管理しています。この番号は 0 から始まるので、client[0] は clclass のインスタンス client の 1 台目のノードです。sync はブレースで囲まれた条件がすべて真になるまでシナリオの実行をストップします。さて、ノードの設定がおわるとノードシナリオで

```

scenario {
    sync {
        msgmatch client[0] "setupdone"
    }
    send client[0] "Hello"
    sync {
        msgmatch client[0] "done"
    }
}

```

図 5: メッセージ交換グローバルシナリオ

記述したように、setupdone が送られています。これを受信すると master は client[0] に Hello を送ります。そして、client[0] から done が送られてくるとシナリオは終了します。

実は master はグローバルシナリオの最後まで行くと実行を終了します。slave は master との通信がとぎれるとその実行していた実験を終了してしまいますので、slave が実験に必要なコマンドを実行しきるまで、master は実行されている必要があります。というわけで、最初の touch の実験では touch が終わるだろう 10 秒間の間は master に生きていてもらうように sleep を入れました。今回はメッセージで終了したタイミングを確認できますので sleep を入れる必要がなかったのです。

5.2.2 実行と確認

では実行してみましょう。でも、さっきすでにインストールしたノードをそのまま使えばいいかと思ったら、method を thru に書き直します。こうするとノードへの OS のインストールを省略します。

実行は先ほどと同じように行います。touch.sc が msg.sc に変わっただけです。pre.sc はそのまま使います。

```

$ cd /home/starbed/bin/
$ master -p starpass pre.sc msg.sc

```

この実験では/tmp/huga にメッセージで受けた文字列を保存しています。グローバルシナリオでは Hello と送りましたので、/tmp/huga を cat など確認して Hello と書いてあれば成功です。

5.3 ネットワークの設定

これまでの実験は全くネットワークを使わないで行っていましたが、そろそろネットワーク関連の実験の準備をしてみましょう。

5.3.1 シナリオを書く

と、その前に、ちょっとシナリオ記述を楽にするための変数の使い方を勉強してみましょう。

SpringOS では sh のように変数を扱えます。先ほど受信したメッセージを変数に保存しましたが、他にはよく使うコマンドなどを変数に入れてしまう方法があります。

図 6 をノード定義の前に書きます。ここではバイナリが置いてあるディレクトリを bindir とし、さらに ifscan というコマンドのパスを pathifscan として定義しています。これらはグローバル変数として利用したいのでノードの定義、そしてグローバルシナリオの外で指定してさらに export します。これで ifscan を使う

```

bindir="/home/starbed/bin/"
pathifscan=bindir+"ifscan"
export bindir
export pathifscan

```

図 6: 変数の扱い

ときにわざわざ ifscan のパスを確認する必要が無くなりました。といっても今回は一度しか ifscan を使わないので、あまりご加護がないかもしれませんが、よく使うコマンドをこのように変数にいれてしまったりするとシナリオ記述が簡単になりますよね。

ではシナリオを書きましょう。ここではネットワークの設定をするので 2 台のノードを用意します。これまで使ってきた clclass のノードを 2 台用意してもいいのですが、ここではこの後の展開も考えてもう一つクラスを定義します。ちなみにクラス定義をそのまま利用する場合は以下のように定義します。このときそれぞれの client は client[0] と client[1] と指定します。

```
nodeset client class clclass num 2
```

まず、clclass を拡張します。図 7 がその内容です。

```

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "cdone"
    }
}

```

図 7: クライアント用クラス定義

シナリオの内容と netif で始まる行が追加されています。netif はネットワークインターフェースでメディアが FastEthernet であることを示しています。ここで指定するメディアは FastEthernet や GigabitEthernet などが使えます。ここでは一行だけ netif という行がありますので、1 つのインターフェースを要求していません。複数個必要な場合は必要な分だけ行を足してください。シナリオの中身では netifit で始まる行がありません。これはネットワークインターフェースの設定を行うときの決まり文句です。SpringOS はノードの各インターフェースとそれが接続されているスイッチのポートを知っていて、その情報を元にネットワークインターフェースへの IP アドレスの設定と VLAN の設定をします。ところが OS によってネットワークインターフェースを識別する順序やデバイス名が異なります。そこで、SpringOS は MAC アドレスで各ネットワークインターフェースを管理しています。そこで MAC アドレスとデバイス名のマッピング情報が必要になります。この情報を ifscan を用いて行っています。ですのでこの行がないと SpringOS はどのインターフェースを設定していいかわからなくなってしまうのです。もちろん OS のデバイス名が前もってわかっているの

あれば、シナリオにそう記述してしまうこともできますが、OSによっては起動するたびに認識する順序が変わってしまうケースもありますので、ちょっと怖いですね。

ここで作ったマッピング情報は `self.netif[番号].rname` で参照できます。番号はやっぱり 0 から始まる数字です。ここでは 1 番目のインターフェースに、そのインターフェースに指定されるべき IP アドレスを `self.netif[0].ipaddr` として参照して設定しています。この情報は `master` が割り当てて、`master` から通知されているものです。これで IP アドレスの指定がおわったので `cdone` というメッセージを `master` に送っています。

さて次に、サーバ用クラスを新たに定義しましょう。図 8 がそれです。

```
nodeclass svclass {
  scenario {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
      netifit pathifscan
      callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
      send "sdone"
    }
  }
}
```

図 8: サーバ用クラス定義

といってもこの時点ではクラス名と設定が終わった後に送るメッセージが違っただけです。

というわけで次の設定にうつります。

SpringOS ではネットワークの設定をノードと同様にクラスとインスタンスの定義で作成します。図 9 にその一例をあげます。

```
netclass ethclass {
  media fastethernet
}
```

図 9: ネットワーククラス定義

ノードは `nodeclass` で定義しましたが、ネットワークは `netclass` で定義します。ここではこのネットワークで利用するメディアを指定しています。

次にノードとネットワークのインスタンスを作ります。ノードはすでに定義したのと同じようにやれば良いです。ネットワークのインスタンスも `nodeset` が `netset` になるだけです。図 10 のようにすれば OK です。

次に作ったネットワークインスタンスにノードのネットワークインターフェースを加えることでネットワークを形成しましょう。ハブにネットワークインターフェースから生えたケーブルを刺していくというイメージです。

図 11 が具体的な記述です。attach をつかって `client` とサーバの一番目のネットワークインターフェースを

```
nodeset client class clclass num 1
nodeset server class svclass num 1
netset ethnet class ethclass num 1
```

図 10: ネットワークへのインターフェースの接続

```
attach client[0].netif[0] ethnet[0]
attach server[0].netif[0] ethnet[0]
```

図 11: ネットワーク作成

ネットワークのインスタンス `ethnet` に接続しています。

最後にグローバルシナリオです。今回は `client` と `server` からの設定終了メッセージを受け取るだけの簡単なものにしましょう。図 12 のように記述します。

```
scenario {
  sync {
    msgmatch client[0] "cdone"
    msgmatch server[0] "sdone"
  }
}
```

図 12: グローバルシナリオ

これでシナリオ記述はおしまいです。全体のシナリオは付録にあります。というわけで実行してみましょう。

さて、実行自体は先ほどとほぼ同じです。シナリオファイルの指定が違うだけです。

```
$ master -p starpass pre.sc net.sc
```

`master` が終了したら実験ノードに IP アドレスの設定がされているはずです。実験用ノードにログインして `ifconfig` で確認してみましょう！ IP アドレスも指定した範囲内にあることを確認してください。そしてどちらかの実験ノードからもう一台の実験ノードへの接続性を `ping` 等で確認してみてください。このとき実験側のネットワークインターフェースについている IP アドレスを使ってください。

6 ちょっと高度なシナリオ記述

さて、ここまでで実験用ノードでのコマンド実行とネットワーク設定、そしてノード同期のしかたを練習しました。実はこれで SpringOS の基本的な使い方が習得できたこととなります。あくまで OS のインストールとネットワーク設定、そしてノードでのコマンド実行が SpringOS の役割なのです。

でも、もうちょっと簡単に実験記述ができるような機能も用意されています。ここでは、このような機能の練習と、これらを使って `netperf` を使った 2 台のノード間の帯域測定を行うシナリオ作成を行ってみま

しょう！

6.1 いろんな機能

では、まず機能群の説明をします。ここでは、以下のことについて勉強しましょう。

- master のオプションとして変数を入力
- ループ制御
- メッセージの内容で動作を変更

6.1.1 master のオプションとして変数を入力

ネットワーク実験というのはちょっとしたパラメータを変化させて何度か実験を繰り返し、その結果を比較してみるといったことが比較的よくなされます。それぞれの実験を行うときにいちいちシナリオファイルを編集するというのは面倒くさいですし、あとでどこをどう編集したかわからなくなってしまったりしたら大変です。一つのシナリオファイルをつかって変数パラメータだけ変更できると、シナリオファイルは一つでよくすっきりしますよね。このような要望を満たすために master はコマンドラインからシナリオ中の変数の初期値を受け取ることができます。これで、master に渡したオプションを記録しておけばどのような実験をしたのが簡単に記録できますよね。

では、実際の使い方です。これまでグローバル変数は以下のように定義していました。

```
pair=1
export pair
```

これで初期値に 1 をもつ pair という変数がシナリオ全体で使えることになりますよね。ここで pair の初期値をコマンドラインからも受け取れるようにしてみましょう。記述を以下のように変更します。

```
assure pair=1
export pair
```

最初の変数定義の前に assure がつきました。これだけです。そしてたとえばこの変数を使ってノードのインスタンスを定義してみます。

```
nodeset client class clclass num pair
```

こうしておくともコマンドラインで指定した数だけノードのインスタンスを作れます。そして変数を定義するときにデフォルトの初期値を 1 として指定しているので、コマンドラインから値が与えられなければ 1 台だけインスタンスが作成されます。

6.1.2 ループ制御

プログラミング言語で非常によく使うループ制御。もちろんできます！ for, loop, while が使えます。基本的には C 言語と同様にかけます。

```
for(i=0;i<pair;i++) {
    ethnet[i].ipaddrange = "192.168."+toString(i)+".0/24"
```

```
}
```

ここでは `i` 番目の `ethnet` に別の IP アドレスレンジを指定しています。for で扱う `i` は数値型なのですが、`ipaddressrange` は文字列型なので、`tostring` で型を変更する必要があります。

6.1.3 メッセージの内容で動作を変更

実験中は、slave で `main loop` をずっとまわしておいて、あるトリガによって master からメッセージ送って、その内容で動作内容を変えたいなんていうことはよくあります。何が起るかわからないネットワーク実験ではむしろこのような形態が自然かもしれません。

```
loop {
  recv val
  msgswitch val {
    "start" {
      call pathnetserv
      send "started"
    }
    "stop" {
      callw "/usr/bin/pkill" "-9" "netserver"
      send "stopped"
    }
  }
}
```

このシナリオでは無制限に loop して、変数 `val` に受け取ったメッセージを保存します。その内容が `start` であれば `pathnetserv` を実行し、`stop` であれば `netserver` を `pkill` で終了しています。もちろん if 文を使って同様のことを行えます。

6.1.4 その他の雑多な事たち

ここでは、そのほかの細かい記述方法を紹介します。

コマンド実行 `callw` を使うとバックグラウンドでのコマンド実行 (wait 有)、`call` を使うとフォアグラウンド実行になります。

時間の導入 `time()` をつかうとエポックタイムを取得できます。たとえば以下のようにするとログファイルに時間を入れられるので、繰り返しの実験をしたときにファイルが上書きされることがありません。

```
epoch=time()
logfile=self.rname+"-"+tostring(epoch)+".log"
```

ディレクトリ移動 ディレクトリを移動します:p

```
chdir workdir
```

ノードの実験用 IP アドレスの参照 netperf ではあるクライアントが対象のサーバに対して通信を行います。サーバが複数ある場合などには、それぞれのクライアントが通信を行うサーバをしないする必要がありますよね。たとえば以下のようにグローバルシナリオに記述すると各クライアントに対応するサーバの IP アドレスを送信できます。

```
for(i=0;i<pair;i++) {
    send client[i] haddr(server[i].netif[0].ipaddr)
}
```

このケースではサーバー台にはクライアント一台のみが割り当てられていますね。i の値は場合によって変更が必要です。

標準出力のリダイレクト ログをリダイレクトしてファイルに保存というのはやっぱりやりたいものですよね。

```
callw "/bin/date" dst > logfile
callw pathnetperf "-H" dst >> logfile
```

ファイルの送信 ログファイルなどをどこかに集めておくのに便利です。FTP サーバはもちろん起動しておいてくださいね。netget を使えば逆にファイルを取得することができます。もちろん wget や ftp を call しても同じです。

```
netput logfile "ftp://starbed:starbed@10.211.55.10/log/clients/"
```

まとめてメッセージ交換 nodeset をつかってインスタンスをまとめて生成しましたが、このグループまとめて同じメッセージ交換をしたいということもよくあります。

```
multisend server "start"
multimsgmatch server "started"
```

ここで server はインスタンスの名前です。server[0], server[1] とするとそれぞれのインスタンスを参照できるんですね。

コメントアウト 文頭に#を！

さてさて、これらの記述をつかって用意した netperf のシナリオを付録においてあります。これを使って実験してみましょう。

6.2 netperf を実行しよう！

このシナリオではログを FTP で実験駆動単位管理用ノードに保存します。そこで ftpd を起動しておく必要があります。FreeBSD の場合は/etc/inetd.conf を編集してコメントアウトされている以下の行を有効にします。最初は行頭に#がついているのでこれを削るだけです。

```
ftp stream tcp nowait root /usr/libexec/ftpd ftpd -l
```

そして/etc/inetd.conf を編集して、以下の行を追加しましょう。

```
inetd_enable=YES
```

このあと inetd を起動します。

```
$ /etc/rc.d/inetd start
```

ftp コマンドをつかって実験駆動単位管理ノードに starbed アカウントで starbed というパスワードでログインできるか確認してみましょう。もちろんユーザ名パスワードは別でもかまいませんし、その方がセキュリティ的には安心でしょう。個々の設定に従ってください。また、ログファイルをアップロードするディレクトリは/home/starbed/log/clients としてありますので、このディレクトリを前もって作成するか、ログの保存先のディレクトリを変更しておいてください。これで準備は完了です。実験を始めてみましょう！

master は以下のように起動します。

```
$ master -p starpass pair=1 pre.sc net.sc
```

ここでは、pair に 1 を代入しています。受け付けられることを明記するためにこのようにしましたが、実はデフォルトを 1 にしてあるので、無くても変わりません。pair=2 などにするときには実験用ノードを必要なだけ予約が必要です。

6.3 実行の確認

さて、この実験では netperf の実行結果をリダイレクトしたものをログファイルとして保存して、実験駆動単位管理用ノードに upload しています。/home/starbed/log/clients 以下に保存されているログファイルを確認してください。ファイル名は“ノード名-エポックタイム.log”となっています。タイムスタンプなどで今回の結果か確認してみてください。内容はたとえば以下のようなものであるはずで

```
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.0.1 (192.168.0.1) port 0 AF_INET
Recv  Send  Send
Socket Socket Message Elapsed
Size  Size  Size  Time  Throughput
bytes bytes bytes secs.  10^6bits/sec

65536 32768 32768 10.00 40.78
```

7 sbpsh を使う

さて、最後に sbpsh の使い方を簡単に勉強しましょう。sbpsh はコマンドラインから様々な操作を受け付けるアプリケーションです。ノードの起動方法の変更や電源断や投入などが行えます。

以下のようにして sbpsh を起動してみましょう。

```
$ sbpsh -r sbpsh.rc
command>
```

sbpsh.rc は設定段階で記述しましたよね。snmpmine が起動しているノードに対しては、

```
command> reboot slave1
command> poweroff slave2
```

などすると再起動や電源断ができます。また WoL 対応ノードで wolagent が起動している場合には、

```
command> poweron slave2
```

とすると電源が入ります。

また、起動するパーティションを変更したい場合は、以下のようにします。slave1 が 1 番目のパーティションに入っている OS で起動します。デフォルトでは 1 は Windows ですね。

```
command> setdiskboot slave1 1
```

8 まとめ

これだけでいろいろな実験ができるはずです。使い方はあなた次第！素敵な使い方を見つけた方は是非 StarBED チームに教えてください！

付録

すべてのシナリオファイルを紹介します。

```
/home/starbed/conf/sc/pre.sc
```

```
rmanager ipaddr "127.0.0.1" port "1234"  
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.0.0/24"  
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/24"  
wolagent ipaddr "172.16.4.253" port "5904" ipaddrrange "172.16.4.0/24"  
fncp ipaddr "172.16.3.101"  
tftpdman ipaddr "172.16.3.101"  
  
nidiskimage "FreeBSD/ni04b.fs"  
nikernel "recover_system/kernel_recover"  
pxeloder "recover_system/pxeboot-nohang"  
setuptimeout total 86400 warm 5
```

```
/home/starbed/conf/sc/touch.sc

user "starbed" "info@starbed.org"
project "starproj"

encd ipaddr "10.211.55.10"

sparenodemain 0
sparenoderatio 100

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"
    scenario {
        callw "/usr/bin/touch" "/tmp/huga"
    }
}

nodeset client class clclass num 1

scenario {
    sleep 10
}
```

```

/home/starbed/conf/sc/net.sc

user "starbed" "info@starbed.org"
project "starproj"

encl ipaddr "10.211.55.10"
ipaddrrange "192.168.100.0/24"

sparenodemain 0
sparenoderatio 100

bindir="/home/starbed/bin/"
pathifscan=bindir+"ifscan"
export bindir
export pathifscan

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "cdone"
    }
}

nodeclass svclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "sdone"
    }
}

```

```
    }  
}  
  
netclass ethclass {  
    media fastethernet  
}  
  
nodeset client class clclass num 1  
nodeset server class svclass num 1  
netset ethnet class ethclass num 1  
  
attach client[0].netif[0] ethnet[0]  
attach server[0].netif[0] ethnet[0]  
  
scenario {  
    sync {  
        msgmatch client[0] "cdone"  
        msgmatch server[0] "sdone"  
    }  
}  
}
```

```

/home/starbed/conf/sc/netperf.sc

user "starbed" "info@starbed.org"
project "starproj"

encl ipaddr "10.211.55.10"
#ipaddrrange "192.168.100.0/24"

sparenodemini 0
sparenoderatio 100

bindir="/home/starbed/bin/"
workdir="/home/starbed/work/"
pathifscan=bindir+"ifscan"
pathnetperf="/usr/local/bin/netperf"
pathnetserv="/usr/local/bin/netserver"
epoch=time()
export bindir
export workdir
export pathifscan
export pathnetperf
export pathnetserv
export epoch

assure pair=1
export pair

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://starbed:starbed@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        logfile=self.rname+"-"+tostring(epoch)+".log"
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "setupdone"
        chdir workdir
    }
}

```

```

        recv dst
        callw pathnetperf "-H" dst > logfile
        sleep 180

        netput logfile "ftp://starbed:starbed@10.211.55.10/log/clients/"

        send "perfdone"
    }
}

nodeclass svclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://starbed:starbed@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        netiffit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "setupdone"
        loop {
            recv val
            msgswitch val {
                "start" {
                    call pathnetserv
                    send "started"
                }
                "stop" {
                    callw "/usr/bin/pkill" "-9" "netserver"
                    send "stopped"
                }
            }
        }
    }
}

netclass ethclass {
    media fastethernet
}

```

```

nodeset client class clclass num pair
nodeset server class svclass num pair
netset ethnet class ethclass num pair

for(i=0;i<pair;i++){
    ethnet[i].ipaddrange = "192.168."+toString(i)+".0/24"
}

for(i=0;i<pair;i++) {
    attach client[i].netif[0] ethnet[i]
    attach server[i].netif[0] ethnet[i]
}

scenario {
    sync {
        multimsghmatch server "setupdone"
        multimsghmatch client "setupdone"
    }

    multisend server "start"

    sync {
        multimsghmatch server "started"
    }

    for(i=0;i<pair;i++) {
        send client[i] haddr(server[i].netif[0].ipaddr)
    }

    sync {
        multimsghmatch client "perfdone"
    }

    multisend server "stop"

    sync {
        multimsghmatch server "stopped"
    }
}

```