SSS-PC ユーザ ガイド第1.0版

株式会社情報科学研究所
http://www.isll.co.jp/

2003年12月2日

目 次

1 SSS-PC の特徴

SSS-PC は IBM-PC 互換機および Sun MicroSystems 社の Ultra60 または Ultra2 およびこれらの互換機で動作する次世代オペレーティングシステムです。 SSS-PC は汎用スケーラブルオペレーティングシステム SSS-CORE の研究開発成果を基に設計・実装されています。

SSS-PC は UNIX や Linux, Windows 2000 のように 1 台の PC またはワークステーション上で複数の仕事を同時に実行するマルチタスク環境を提供します。 また、複数台の PC や ワークステーションが LAN によって結合されている環境においては、従来のインターネットやイントラネットの分散処理環境に加えて、マシン群をあたかも 1 台の高性能汎用並列計算機と見なせる環境を提供します。 この場合に、専用並列計算機に見られる特殊なメモリコントローラや特殊な通信用ハードウェア等を一切必要としません。 この環境の上では、1 台のワークステーションにおけるマルチタスク環境と同様に、複数の並列アプリケーションを同時に実行させることが可能です。

SSS-PC には実行中のアプリケーションプログラムを他のマシンに移送して実行する、マイグレーション機能が組み込まれています。 並列アプリケーションの 1 タスク (プロセスと同等) のマイグレーションも可能です。SSS-PC は LAN 内にある PC やワークステーションを自由に選んでクラスタを構成することが可能ですが、マイグレーション機能を組み合わせると、分散計算環境と同様に 1 台のマシンのみを停止させることや、新たにマシンを起動してクラスタに加えることが任意の時点で行えます。 この時にマイグレーションして他のノードに移ったアプリケーションは何の影響もなく継続実行されます。 クラスタを構成するのマシンの 1 台が突然故障により停止した場合でも、その影響が他のマシンに及ぶことはありません。

SSS-PC Ver. 2.x はネットワークブートによって起動されるため、必要に応じて既存のオペレーティングシステムと切り替えて使用することも可能です。

2 インストール

配布 CD-ROM は、FreeBSD 4.8R で動作する PC にインストールし、3 節に説明する tftp サーバもしくは extfs サーバとして利用することを前提に収録されたものです。 他の OS もしくはバージョンでは動作いたしませんので、ご注意下さい。

配布 CD-ROM の内容を示します。

- 配布 CD-ROM の内容

/doc	:	ドキュメントを収めたディレクトリ
/floppy	:	SSS-PC ノードをネットワーク越しに起動する EtherBoot FD のイメージファイルを収めたディレクトリ
/install.sh	:	インストール用シェルスクリプト
/usr.sss	:	/usr/sss にインストールされる SSS-PC サーバのファイルシステムを収めたディレクトリ
/usr.local.sss	:	/usr/local/sss にインストールされる SSS-PC のクロス開発環境を収めたディレクトリ

インストールはインストール用シェルスクルプト install.sh を用いて行います。

- CD-ROM を /cdrom にマウントし install.sh を起動します。

```
# mount -rt cd9660 /dev/cd0a /cdrom
# /cdrom/install.sh
```

  - シェルスクルプトのメッセージに従いインストールを進めて下さい。

- CD-ROM のマウントポイントは必ず /cdrom として下さい。
- インストール先を /usr/sss もしくは /usr/local/sss 以外とした場合、各々 /usr/sss, /usr/local/sss にシンボリックリンクが作成されます。

デフォルトのインストール先を示します。　アンインストールする場合は、これらのパスを手動で消去して下さい。

- デフォルトのインストール先

/usr/sss	:	SSS-PC のファイルシステム
/usr/local/sss	:	SSS-PC のクロス開発環境

3　SSS-PC ノードの設定

SSS-PC Ver.2.x はネットワークブートによって起動されるため、ブートイメージ (カーネル) と外部委託ファイルシステムを提供するサーバが必要となります。
外部委託ファイルシステムが SSS-PC の独自仕様となっているため、同ファイルシステムを利用する場合、extfs サーバとして利用可能な OS は、現在のところ FreeBSD 4.8R [1] と Solaris 8 [2]のみです。　外部委託ファイルシステムを利用しない場合については、ブートイメージ (カーネル) のロードに用いる bootpd(8) と tftpd(8) および arp(8) もしくは rarpd(8) が、動作する様々な OS をサーバとして利用することができます。

本章では FreeBSD をサーバとした場合の SSS-PC ノードの設定方法について説明します。

混乱を避けるため本ドキュメントでは、ネットワークブートと tftp によるファイル転送を提供するサーバを tftp サーバ、外部委託ファイルシステムを提供するサーバを extfs サーバと称します。

3.1　ネットワークブートの概要

ネットワークブートを行うための tftp サーバー設定について説明します。　SSS-PC ノードのネットワークブートの流れは、次のようになっています。

1. EtherBoot が RARP によって自ノードの IP アドレスを取得する。
2. EtherBoot が BOOTP によって tftp サーバ上の /etc/bootptab に記載されるブートパラメタを取得する。
3. EtherBoot がブートパラメタに指定される SSS-PC のブートイメージ (カーネル) を TFTP を用いロードする。
4. SSS-PC カーネルが RARP によって自ノードの IP アドレスを取得する。
5. SSS-PC カーネルが BOOTP によって tftp サーバ上の /etc/bootptab に記載されるブートパラメタを取得する。
6. SSS-PC カーネルがブートパラメタに指定されるクラスタ構成定義ファイルを TFTP を用い取得する。
7. SSS-PC カーネルは取得したクラスタ構成定義ファイルに基づき起動処理を行う。

以上の処理を実現するため、SSS-PC ノードを起動する PC については EtherBoot FD の作成が、tftp サーバについては RARP(/etc/hosts,/etc/ethers)、BOOTP(/etc/bootbtab)、TFTP 等の設定が各々必要となります。
設定を行う前に SSS-PC を起動する PC のホスト名と IP アドレスを決め、NIC(Network Interface Card) のメーカー／型番／チップセットと MAC アドレス (Ether アドレス) を調べておきます。

[1] i386 アーキテクチャのみ、alpha,pc98 はサポートしておりません。
[2] SPARC アーキテクチャのみ、Intel 版はサポートしておりません。

便宜上、本稿では次表のホスト名、IP アドレス、MAC アドレスが用いられるものと仮定します。　また、ネットマスクは 255.255.255.0、デフォルトゲートウェイは 192.168.1.1 であるものと仮定します。

	ホスト名	IP アドレス	MAC アドレス
tftp/extfs サーバ	sss-svr	192.168.1.100	5e:4d:3c:2b:1a:00
PC1	sss-pc1	192.168.1.101	5e:4d:3c:2b:1a:01
PC2	sss-pc2	192.168.1.102	5e:4d:3c:2b:1a:02
PC3	sss-pc3	192.168.1.103	5e:4d:3c:2b:1a:03
PC4	sss-pc4	192.168.1.104	5e:4d:3c:2b:1a:04

3.2 RARP に関する設定

SSS-PC ノードから要求される RARP リクエストに対し IP アドレスを返すため、rarpd(8) の設定を行います。　rarpd(8) が読み込む MAC アドレスと IP アドレスは、/etc/ethers と /etc/hosts に設定します。

- /etc/ethers の設定

```
5e:4d:3c:2b:1a:01 sss-pc1
5e:4d:3c:2b:1a:02 sss-pc2
5e:4d:3c:2b:1a:03 sss-pc3
5e:4d:3c:2b:1a:04 sss-pc4
```

- /etc/hosts の設定

```
192.168.1.101 sss-pc1
192.168.1.102 sss-pc1
192.168.1.103 sss-pc2
192.168.1.104 sss-pc3
```

/etc/ethers と /etc/hosts を設定した後、rarpd(8) を起動します。

- rarpd(8) の起動

```
# rarpd
```

また、tftp サーバがブートされる際、rarpd(8) が起動するよう /etc/rc.conf を設定します。

- /etc/rc.conf の設定

```
rarpd_enable="YES"
rarpd_flags=""
```

3.3 BOOTP に関する設定

/etc/bootptab の設定を行います。　/usr/sss/etc/bootptab.sample を /etc/bootptab にコピーし、'.default:' 行以下を変更します。　変更が必要となる点は、'.default:' に設定されているネットマスク値 'sm='、各ノードのホスト名 ('sss-pc1:','sss-pc2:','sss-pc3:','sss-pc4:') と、その MAC アドレス 'ha='、IP アドレス 'ip=' です。

- /etc/bootptab の設定

```
.motd:\
:T184="^[[37m+---------------+":\
:T186=" sss-svr - bootpd":\
:T190="+---------------+^[[37m":
```

```
# label:server:gateway:filename:passwd:flags:cmdline

# :tc=.motd:\
.imagemenu:\
:T184="default bianry image=(A)":\
:T128=E44574680000:\
:T133="-c0 -b38400 Hello SSS-PC":\
:T160="timeout=10:default=192:":\
:T192="^[[37mSSS-PC^[[37m:::/lpc12.20031202A:":\
:T195="^[[37mlocal hda^[[37m:::/dev/hda:":\
:T196="^[[37mdos^[[37m:::/dos.bin:":\
:T150="/boot/conf/ssspc.conf":

## :T133="-extfsip 192.168.1.100 -extfsport 5002":
## :T196="^[[37mlocal floppy^[[37m:::/dev/fda:":
## :T197="^[[37mdos^[[37m:::/dos.bin:":

.default:\
:tc=.imagemenu:\
:ht=ethernet:\
:sm=255.255.255.0:\
:hd=/:\
:hn:\
:to=-18000:\
:T132=00400000:\
:T133="Hello SSS-PC":

.default-old:\
:ht=ethernet:sm=255.255.255.0:hd=/:hn:to=-18000:

sss-pc1:\
:ha=5e4d3c2b1a01:ip=192.168.1.101:bf=cluster.pc:tc=.default:

sss-pc2:\
:ha=5e4d3c2b1a02:ip=192.168.1.102:bf=cluster.pc:tc=.default:

sss-pc3:\
:ha=5e4d3c2b1a03:ip=192.168.1.103:bf=cluster.pc:tc=.default:

sss-pc4:\
:ha=5e4d3c2b1a04:ip=192.168.1.104:bf=cluster.pc:tc=.default:
```

- 'bf=' に設定される 'cluster.pc' は、/usr/sss に置かれるクラスタ構成定義ファイルのファイル名です。　一般的なネットワークブートでは 'bf=' にブートイメージファイル名を設定しますが、SSS-PC では、クラスタ構成定義ファイル名である点に注意して下さい。
- SSS-PC のブートイメージ (カーネル) ファイル 'lpc12.20031202A' は、'T192:' に設定されています。
- extfs サーバをネットブートに用いる tftp サーバ以外のマシンに指定する場合は、'T133:' に設定します。
- '^[[37' の '^[' は 'Ctrl+[' で入力されるコントロールシーケンスである点に注意して下さい。

bootpd(8) は、通常 inetd(8) から起動されます。　/etc/inetd.conf を変更し、inetd(8) を 'kill -HUP' します。　'kill -HUP' を行う前に TCP wrapper (/etc/hosts.allow,/hosts,deny) の設定も確認して下さい。

- /etc/inetd.conf の設定

```
bootps  dgram   udp     wait    root    /usr/libexec/bootpd     bootpd
```

- inetd(8) を 'kill -HUP' する。

```
# kill -HUP `cat /var/run/inetd.pid`
```

3.4 TFTP に関する設定

インストールされた /usr/sss を tftpd(8) のルートディレクトリに設定します。

tftpd(8) は、通常 inetd(8) から起動されます。 /etc/inetd.conf を変更し、inetd(8) を 'kill -HUP' します。'kill -HUP' を行う前に TCP wrapper (/etc/hosts.allow,/hosts,deny) の設定も確認して下さい。

- /etc/inetd.conf の設定

```
tftp    dgram   udp     wait    root    /usr/libexec/tftpd      tftpd -l -s /usr/sss
```

- inetd(8) を 'kill -HUP' する。

```
# kill -HUP `cat /var/run/inetd.pid`
```

次にクラスタ構成定義ファイル /usr/sss/cluster.pc を設定します。 クラスタ構成定義ファイルは SSS-PC 独自の設定ファイルです。

- クラスタ構成定義ファイルの書式を示します。

```
{"name",macaddrhigh,macaddrlow,ipaddr,dsp_f,d_pos,{connect_f}},
```

name	:	ホスト名
macaddrhigh	:	MAC アドレスの上位 4 バイト
macaddrlow	:	MAC アドレスの下位 2 バイト (MSB へ 16 bits シフト)
ipaddr	:	IP アドレス (HEX 表記)
dsp_f	:	コンソールにメッセージ表示画面を立ち上げるか否かを示すフラグ
d_pos	:	リザーブ ('0' として下さい)
connect_f	:	左から 1 列目: 100Mbps のネットワークインターフェースを搭載している場合は '1'
		2 列目: 1000Mbps のネットワークインターフェースを搭載している場合は '1'
		3 列目: リザーブ ('0' として下さい)
		4 列目: リザーブ ('0' として下さい)

  - 先頭が '#' で始まる行はコメントとみなします。
  - 先頭行のエントリから順に遠隔ノード番号 (PNID:Parallel node ID) 0, 1, 2 ... が割り振られます。 遠隔ノード番号 0 はサーバとみなされますが、tftp サーバは各ノードで、eextfs サーバは /etcc/bootptab で、各々設定されるため、システム的な意味はありません。
  - 以下の例では sss-pc1 の遠隔ノード番号が 1、sss-pc2〜4 は 2〜4 となります。
  - 最後の行は、以下の例に示す通り '{ "LastNode", ... 0}}.' として下さい。
  - dsp_f フラグの設定は通常以下の例に示す通りサーバと LastNode を除き '1' として下さい。

- /usr/sss/cluster.pc の設定

```
#  name       macaddrhigh macaddrlow  ipaddr      dsp_f d_pos connect_f
 { "sss-svr", 0x5e4d3c2b, 0x1a000000, 0xc0a80164,   0,    0, { 1, 0, 0, 0} },
 { "sss-pc1", 0x5e4d3c2b, 0x1a010000, 0xc0a80165,   1,    0, { 1, 0, 0, 0} },
 { "sss-pc2", 0x5e4d3c2b, 0x1a020000, 0xc0a80166,   1,    0, { 1, 0, 0, 0} },
 { "sss-pc3", 0x5e4d3c2b, 0x1a030000, 0xc0a80167,   1,    0, { 1, 0, 0, 0} },
 { "sss-pc4", 0x5e4d3c2b, 0x1a040000, 0xc0a80168,   1,    0, { 1, 0, 0, 0} },
 { "LastNode",0x00000000, 0x00000000, 0x00000000,   0,    0, { 0, 0, 0, 0} },
```

3.5 EtherBoot FD の作成

SSS-PC ノードをネットワーク越しに起動する EtherBoot FD を作成します。

- 起動に用いるイメージファイルは NIC(Network Interface Card) のメーカー／型番／チップセットによって異なります。

NIC	イメージファイル
Intel EtherExpress PRO/100B (82557, 82558)	ethboot-ee100.img
3Com 3c905	ethboot-3c905.img

- 配布 CD-ROM の /floppy から適切なイメージファイルを選び、dd(1) を用いて FD に書き込みます。

```
# mount -rt cd9660 /dev/cd0a /cdrom
# dd if=/cdrom/floppy/ethboot-ee100.img of=/dev/rfd0 bs=36b
```

3.6 外部委託ファイルシステムの起動

外部委託ファイルシステム freebsd_fs は SSS-PC 独自仕様の簡易なネットワークファイルシステムです。
SSS-PC の各ノードは、ネットブートに用いられた tftp サーバ、もしくは /etc/bootptab の 'T133:' に設定されたサーバ (3.3 節参照) を外部委託ファイルシステムのサーバ (extfs サーバ) と見なしコネクションを図ります。
extfs サーバー側では次の要領で freebsd_fs を動作させておきます。

- freebsd_fs の起動

```
# chroot /usr/sss /freebsd/boot/freebsd_fs
```

  - freebsd_fs には、chroot を行う機能が含まれていないため、chroot(8) と併用します。
  - X Windows 上で新しいコンソール (xterm 等) を開いて実行して下さい。

- /etc/rc.local 等で自動起動させる例

```
# chroot /usr/sss /freebsd/boot/freebsd_fs > /dev/null 2>&1 &
```

3.7 トラブルシューティング

3.5 節で作成した EtherBoot FD を SSS-PC を起動する PC に挿入、PC の電源をオン (又はリセット) し、起動してみます。
SSS-PC の起動／停止方法については 4 節を参照して下さい。

もし、旨く立ち上がらない場合は tcpdump(1) や bootpd(8) を用い問題点を探すとよいでしょう。

- tcpdump(1) を用い SSS-PC のノードとなる PC からリクエストが送られているか確認する。

```
# tcpdump port 67
```

- bootpd(8) をデバッグモードで立ち上げ、 /etc/bootptab の情報が受け渡されているか確認する。

```
# bootpd -d4
```

  - /etc/inetd.conf の bootbs をコメントアウトし inetd(8) を 'kill -HUP' しておきます。

4 SSS-PC の起動と停止

4.1 SSS-PC の起動方法

3.5 節で作成した EtherBoot FD を SSS-PC を起動する PC に挿入、PC の電源をオン (又はリセット) します。

以下に示す通り起動時には、/etc/bootptab の設定 (3.3 節参照) に基づき、ロード可能なブートイメージ (カーネル) の一覧が表示され、'A'〜'C' キーを押すことでブートイメージ (カーネル) を選択することができます。何も押さなければデフォルトに設定されている SSS-PC のブートイメージ (カーネル) lpc12.20031202A がロードされます。

- ブートローダーのメッセージ

```
ROM segment 0x0800 length 0x8000 reloc 0x9400
Etherboot 5.0.2 2001-06-23 (GPL) FreeBSD for [EEPRO100]
Found Intel EtherExpressPro100 at 0xC800, ROM address 0x9800
Probing...[EEPRO100]Ethernet addr: 5E:4D:3C:2B:1A:01
Searching for server (DHCP)...
Me: 192.168.1.101, Server: 192.168.1.100

default bianry image=(A)

List of available boot images:

    A) SSS-PC
    B) /dev/hda
    C) dos

Select: SSS-PC                                  ← 10 秒以内に'A'〜'C' を選択することができる
Loading 192.168.1.100:/lpc12.20031202A (ELF entry:0xC0000000 loading:0x00400000)...done
```

ブート時にグラフィックモードの選択を求められます。　CRT が 1280x1024 表示に対応可能であれば 'y' キーを押して下さい。　1280x1024 表示に対応できない場合は、'SPACE' キーを押します。

- グラフィックモードの選択

```
Do you use 1280x1024 graphics mode? (y/n):n   ← CRT が 1280x1024 表示対応である場合は、'y'
                                                 表示できない場合は、'SPACE' キー.
```

SSS-PC が起動されると、左上にメッセージ表示画面、右下に NikoMon Shell のコンソールが立ち上がります。

- NikoMon Shell の起動メッセージ

```
Hello SSS world!! SSS-PC Ver.2.3 (Tue Nov 11 11:20:22 JST 2003)
NikoMon ready
:-)
```

4.2 SSS-PC の停止方法

NikoMon Shell のコマンドプロンプト':-)' に対し、reboot コマンドを入力することで SSS-PC を停止することができます。

- reboot コマンドの実行

```
:-)reboot
```

メッセージ表示画面に白抜きで次のメッセージが表示された後、PC の電源を OFF (又は、リセット) して下さい。

- SSS-PC の停止を示すメッセージ

```
Reboot and Select proper Boot device
or Insert Boot Media in selected Boot device
```

  - メッセージ表示画面をスクロールさせていた場合等、メッセージが表示されない場合があります。　メッセージが表示されない場合は、reboot コマンド入力後 10 秒程度待って PC の電源を OFF (又は、リセット) して下さい。

5 NikoMon Shell

SSS-PC の操作は、NikoMon Shell と呼ばれるモニタプログラムから行います。　本節では NikoMon Shell の使い方について簡単に説明します。

5.1 組み込みコマンド

NikoMon Shell には、**組み込みコマンド**と呼ばれる基本的な操作に必要なコマンドが組み込まれています。

- NikoMon Shell の組み込みコマンド

```
:-)command -
node1: argc=2 argv=[command -]
      ipconf       ipstat     e100stat         icon         scrn       colors
       clear         cmap         recv         send        close         open
      status        start       submit         load          cat         echo
          cd          pwd          env          arg       putenv       lstftp
       extfs   csmfileclr       csmclr      csmsave      csmload      objload
    mem2code      taskque      freemem      debugos       reboot     prereset
     regdump      stkdump          bye         stty       lbreak       memset
     memdump     taskdump       resume         stop         kill       signal
      myname       oblist           bg           fg         jobs           ps
     command     pobjload        pload      apstart       pstart          psh
         msh      nikomig
:-)
```

個々のコマンドについては、**8 NikoMon Shell コマンドリファレンス** を参照して下さい。

5.2 入力操作

5.2.1 特殊文字と Parse 機能

NikoMon Shell では、次の文字を特殊文字として扱います。

- NikoMon Shell の特殊文字

```
! @ $ ^ = \ ‘ ; : " ’ , < > ? SPACE TAB
```

また、次の条件に一致する文字列も特殊な扱いを受けます。

- 先頭がシャープ '#' の場合は、コメントとして扱われます。

```
:-)#aaa                          ← 先頭が '#' の場合は無視されます.
:-) #aaa
node1: argc=1 argv=[#aaa]
"#aaa" not found
:-)
```

- 数字に挟まれた '–' は、連番への展開を意図します。

```
:-)aaa 1 -- 8
node1: argc=9 argv=[aaa 1 2 3 4 5 6 7 8]
"aaa" not found
:-)
```

- '$' は、環境変数に置き換えます。

```
:-)putenv foo="hoge"
node1: argc=2 argv=[putenv foo=hoge]
:-)echo $foo
node1: argc=2 argv=[echo hoge]
argv[1]=4"hoge"
hoge
:-)
```

これらの文字をコマンドの引数として入力する場合は、ダブルクォーテション '"' もしくはシングルクォーテション "'" で括ります。

- 特殊文字の入力例

```
:-)lynx "http://www.example.jp"
:-)lynx http":"//www.example.jp
:-)dig '@192.168.1.100'
:-)dig '@'192.168.1.100'
```

5.2.2 History 機能

Ctrl+P で、直前に実行したコマンドをコマンドプロンプトへ呼び出すことができます。 Ctrl+P を繰り返し入力することで更に過去に遡り実行したコマンドを呼び出すことができます。

5.2.3 Emacs ライクなキーバインド

コマンドプロンプトに入力している文字列は Emacs ライクなキーバインドで編集することができます。

5.2.4 コンソール画面のスクロール

コンソール画面には、左上方にシステムメッセージ表示領域、右下方に NikoMon 端末領域があります。 システムメッセージ表示領域も NikoMon 端末も後方スクロールなどの操作ができます。
スクロール対象画面はシステムメッセージ表示領域と NikoMon 端末の 2 つがあり、Meta-K によって切替えます。
1 行ごとのスクロールは、前方へは Meta-N、後方へは Meta-P で行ないます。 画面単位でのスクロールは、前方へは Ctrl-V、後方へは Meta-V で行ないます。 スクロールを終了し、NikoMon のコマンドラインに戻るには、Meta-B をおします。

- コンソール画面のスクロール
  Meta-N : 1 行前方へスクロール
  Meta-P : 1 行後方へスクロール
  Meta-B : スクロールの終了
  Ctrl-V : 1 画面前方へスクロール
  Meta-V : 1 画面後方へスクロール
  Meta-K : スクロール対象画面の切替

5.3 プログラムのロード

組み込みコマンドを除くコマンド (プログラム) は、load コマンド等を用いて NikoMon Shell 上にロードし実行することができます。 NikoMon Shell の tftp サーバに対するファイルアクセスは全て tftp を介して行われます。 load コマンドは、tftp を介し、tftp サーバ上の /usr/sss 以下から引数に指定されたファイルをメモリへロードします。

- load コマンドの実行例

```
:-)load /ssspc/demo/hellow_world
node1: argc=2 argv=[load /ssspc/demo/hellow_world]
load: a_magic=457f a_text=10101 a_data=0 a_bss=0 a_entry=30002
p_offset = 0x0, p_memsz = 0x21110
pn = 0x22
before memcpy(0x40000000,0x60008000,135440)
p_offset = 0x22000, p_memsz = 0x16870
pn = 0xa
before memcpy(0x40000000,0x6002a000,38240)
load: entry=10000074
 text addr=10000 npages=22 phypage=1480
 data addr=10022 npages=a phypage=16e0  bss addr=1002c npages=d phypage=ffffffff
:-)
```

  - load コマンドの引数に指定されるファイル名は、tftp サーバの /usr/sss を root '/' とするものである点に注意して下さい。 /ssspc/demo/hellow_world は、tftp サーバー上の /usr/sss/ssspc/demo/hellow_world に相当します。

load コマンドによりロードされたプログラムが freebsd_fs によりマウントされたファイルシステムを解釈することはあっても、NikoMon Shell 自身はファイルシステムを解釈しません。 NikoMon Shell における全てのファイルアクセスは tftp を介して行われる点に注意して下さい。

5.4 プログラムの実行

メモリにロードされたプログラムは組み込みコマンドと同様に扱われます。

- command コマンドで使用可能なコマンドの一覧を見てみます。

```
:-)command -
node1: argc=2 argv=[command -]
hellow_world       ipconf       ipstat     e100stat         icon         scrn
      colors        clear         cmap         recv         send        close
        open       status        start       submit         load          cat
        echo           cd          pwd          env          arg       putenv
      lstftp        extfs   csmfileclr       csmclr      csmsave      csmload
     objload     mem2code      taskque      freemem      debugos       reboot
    prereset      regdump      stkdump          bye         stty      lbreak
      memset      memdump     taskdump       resume         stop         kill
```

```
     signal       myname       oblist           bg           fg         jobs
         ps      command     pobjload        pload      apstart       pstart
        psh          msh      nikomig
:-)
```

  - hellow_world がコマンドの一覧に追加されていることがわかります。

- '-' オプション無しで command コマンドを実行するとロードされたプログラムのみの一覧を見ることができます。

```
:-)command
node1: argc=1 argv=[command]
hellow_world
:-)
```

- hellow_world コマンドの実行例

```
:-)hellow_world
node1: argc=1 argv=[hellow_world]
:-)Hellow World!!


 !! Task(#1cb2c:0x4001cb2c) Finished !! (cmd="hellow_world",ret=15)
```

5.5 プログラムの削除

ロードされたプログラムは kill コマンドを用いてメモリから削除することができます。

- kill コマンドの実行例

```
:-)command
node1: argc=1 argv=[command]
hellow_world
:-)kill hellow_world
node1: argc=2 argv=[kill hellow_world]
kill(hellow_world)=0,(obj:-1,cmd:0,code:-1)
:-)command
node1: argc=1 argv=[command]
:-)
```

5.6 環境変数

NikoMon Shell における環境変数の設定は putenv コマンド、その表示には env コマンドを用います。

- putenv, env コマンドの実行例

```
:-)putenv PATH="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon"
node1: argc=2 argv=[putenv PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon]
:-)env
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon
```

  - 特殊文字 (5.2.1 節参照) が含まれている文字列は ”’ で括る必要がある点に注意して下さい。
  - 例に示した PATH 環境変数は NikoMon Shell 自身が必要とするものではありません。 NikoMon Shell から起動されるアプリケーションから getenv(”PATH”) された場合に引き渡されるものです。

設定すべき主な環境変数な環境変数を示します。

- 設定すべき主な環境変数

```
USER HOME UID GID PATH LOGHOST DISPLAY TERM TERMCAP
```

  - 各環境変数は一般的な UNIX 系 OS のものと同じ意味を持ちます。
  - TERM 環境変数と TERMCAP 環境変数については /ssspc/termcap (ttfp サーバ上は /usr/sss/ssspc/termcap) に、nikomon, vt100, kterm 用の設定ファイルが用意されています。　次項に説明する submit コマンドを用いて読み込んで下さい。　コンソールには nikomon を用いて下さい。

5.7 submit ファイル

submit コマンドは、コマンドプロンプトに対しコマンドをキーボード入力する代わりに tftp サーバー上に予め用意されたファイルから読み込むものです。 1em このようなファイルを **submit ファイル**と呼びます。
前項の環境変数や次項のネットワークパラメタの設定等に用います。

- 環境変数設定用 submit ファイル (rc.sample) の例

```
putenv USER=user1
putenv HOME=/ssspc/user1
putenv UID=1001
putenv GID=20
putenv PATH="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon"
putenv LOGHOST="192.168.1.100"
putenv DISPLAY="192.168.1.100:0"
putenv TERM=nikomon
putenv TERMCAP="console|nikomon:bl=^G:cm=5\E[%i%d;%dH:so=2\E[7m:se=2\E[m:
                ce=3\E[K:cl=50\E[H\E[J:up=\E[A:do=\E[B:nd=\E[C:le=\E[D:"
env
command
```

  - TERMCAP 変数はドキュメント上に分割していますが、1 行で記載して下さい。

- rc.sample を submit コマンドで読み込む例

```
:-)submit /ssspc/rc.sample
node1: argc=2 argv=[submit /ssspc/rc.sample]
node1: argc=2 argv=[putenv USER=user1]
node1: argc=2 argv=[putenv HOME=/ssspc/user1]
node1: argc=2 argv=[putenv UID=1001]
node1: argc=2 argv=[putenv GID=20]
node1: argc=2 argv=[putenv PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon]
node1: argc=2 argv=[putenv LOGHOST=192.168.1.100]
node1: argc=2 argv=[putenv DISPLAY=192.168.1.100:0]
node1: argc=2 argv=[putenv TERM=nikomon]
node1: argc=2 argv=[putenv TERMCAP=console|nikomon:bl=^G:cm=5\E[%i%d;%dH:so=2\E[7m:se=2\E[m:
                                    ce=3\E[K:cl=50\E[H\E[J:up=\E[A:do=\E[B:nd=\E[C:le=\E[D:]
node1: argc=1 argv=[env]
USER=user1
HOME=/ssspc/user1
UID=1001
GID=20
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon
LOGHOST=192.168.1.100
DISPLAY=192.168.1.100:0:0
TERM=nikomon
TERMCAP=console|nikomon:bl=^G:cm=5\E[%i%d;%dH:so=2\E[7m:se=2\E[m:ce=3\E[K:
                        cl=50\E[H\E[J:up=\E[A:do=\E[B:nd=\E[C:le=\E[D:
```

```
node1: argc=1 argv=[command]
        sh lynx
:-)
```

5.8 ネットワークパラメタの設定

SSS-PC ノードは、ブート時に tftp サーバより RARP を用いて IP アドレスを取得しますが、ネットワークに関するその他のパラメータは設定されません。
ネットマスクやデフォルトルート等のパラメタは、ipconf コマンドを用いて設定します。

- ネットワークパラメタの設定例

```
ipconf setval 0 netmask 255.255.255.0
ipconf setval 0 defaultgateway 192.168.1.1
ipconf setstr 0 hostname "sss-pc1"
ipconf setstr 0 ipdomain "example.jp"
ipconf setstr 0 ypdomain "YP-Example-Domain"
```

5.9 外部委託ファイルシステムのキャッシュ制御

ノード数が数十、数百と増えた場合、外部委託ファイルシステムに対するアクセスがオーバーヘッドとなりうることから、SSS-PC ではファイルサイズが 64KB 以下のファイルをトータル 4MB までメモリ上にキャシュします。
キャシュされたファイルはプログラムから syscall() により、もしくは 'extfs kill' コマンドを用いて明示的に削除しない限り、書き換え (または削除) られることはありません。
キャシュファイルのトータル が 4MB を越えた場合、キャシュは行われなくなります。

'extfs dump' コマンドによりキャシュされているファイルの一覧を得ることができます。

- 'extfs dump' コマンドの実行例

```
:-)extfs dump
node1: argc=2 argv=[extfs dump]
No1: fname:/etc/hosts
  status:----V ref=0 size=1085 st_f=1 cl_tsk=0x0 nxt=0x0 prv=0x0
No2: fname:/etc/resolv.conf
  status:----V ref=0 size=44 st_f=1 cl_tsk=0x0 nxt=0x0 prv=0x0
No3: fname:/etc/host.conf
  status:----V ref=0 size=205 st_f=1 cl_tsk=0x0 nxt=0x0 prv=0x0
:-)
```

キャシュされているファイルを消去する場合は 'extfs kill' コマンドを用います。

- 'extfs kill' コマンドの実行例

```
:-)extfs kill /etc/host.conf                    ← 'extfs kill' を実行
node1: argc=3 argv=[extfs kill /etc/host.conf]
:-)extfs dump                                   ← 'extfs dump' で確認
node1: argc=2 argv=[extfs dump]
No1: fname:/etc/hosts
  status:----V ref=0 size=1085 st_f=1 cl_tsk=0x0 nxt=0x0 prv=0x0
No2: fname:/etc/resolv.conf
  status:----V ref=0 size=44 st_f=1 cl_tsk=0x0 nxt=0x0 prv=0x0
:-)
```

キャシュが働いているとアプリケーションの設定ファイルをカット&トライで調整している場合等、アプリケーションを起動する度に 'extfs kill' コマンドで設定ファイルを消去しなくてはなりません。
'extfs cache' コマンドを用いキャシュ機能を止めておくことができます。

- 'extfs cache' コマンド
  extfs cache : キャシュのステータス (ON/OFF) を表示します。
  extfs cache stop : キャシュ機能を停止します。
  extfs cache resume : キャシュ機能を再開します。

- 'extfs cache' コマンドの実行例

```
:-)extfs cache
node1: argc=2 argv=[extfs cache]
extfs: extfs cache is in operation:
:-)extfs cache stop                    ← キャシュ機能を停止
node1: argc=3 argv=[extfs cache stop]
extfs: extfs cache is stoped:
:-)extfs cache
node1: argc=2 argv=[extfs cache]
extfs: extfs cache is at a standstill:
:-)extfs cache resume                  ← キャシュ機能を再開
node1: argc=3 argv=[extfs cache resume]
extfs: extfs cache is resumed:
:-)extfs cache
node1: argc=2 argv=[extfs cache]
extfs: extfs cache is in operation:
:-)
```

5.10 NikoMon Shell へのリモートアクセス

NikoMon Shell は、SSS-PC のコンソールから利用する他、telnet(1) を用いてリモートアクセスすることも可能です。 ただし、コンソールから利用する場合に比べ、いくつかの機能が制限されています。

- SSS-PC ノード (192.168.1.101) に telnet でアクセスします。

```
# telnet 192.168.1.101
telnet 192.168.1.101
Trying 192.168.1.101...
Connected to sss-pc1.
Escape character is '^]'.

Hello SSS world!! SSS-PC Ver.2.3 (Tue Nov 11 11:20:22 JST 2003)
NikoMon ready, shell_path=0x105 conn_id=4
:-)
```

- bye コマンドで telnet セッションをクローズします。

```
:-)bye
node1: argc=1 argv=[bye]
Connection closed by foreign host.
#
```

5.11 並列処理の実行

複数の SSS-PC ノードに並列処理を実行させる場合は、load コマンドの代わりに pload コマンドを用いてプログラムをロードし、pstart コマンドを用いて実行します。 ただし、プログラムは並列処理用に作成されたものである必要があります。

- NPB プログラム mg.W.4(6.1 節参照) を SSS-PC ノード 1〜4 (sss-pc1〜4) へロードする例

```
:-)pload 1 2 3 4 : /ssspc/NPB2.3_bin/mg.W.4
```

- ロードした mg.W.4 を SSS-PC ノード 1〜4 で実行する例

```
:-)pstart 1 2 3 4 : mg.W.4 -v
```

  - 6.1 節に mg.W.2 のロード／実行結果があります。

6 アプリケーション

6.1 NAS Parallel Benchmarks

NAS Parallel Benchmarks(NPB) は、NASA Ames Research Center で開発された、並列コンピュータのためのベンチマークです。NPB は、5 つの Parallel Kernel Benchmarks と 5 つの Parallel CFD (Computational Fluid Dynamics) Application Benchmarks から構成されており、並列コンピュータの実効性能を知る上で、権威あるベンチマークの 1 つです。

- NBP については、次の URL を参照して下さい。

```
http://www.nas.nasa.gov/Software/NPB/
```

- NPB のプログラムは、次のディレクトリに用意されています。

```
/usr/sss/ssspc/NPB2.3_bin
```

- プログラムのファイル名は次のように定義されています。

xx.Y.N : xx : 問題種別
- ep : 乗算合同法による一様乱数、正規乱数の生成
- mg : 簡略化されたマルチグリッド法のカーネル
- cg : 正値対称な大規模疎行列の最小固有値を求めるための共役勾配法
- ft : FFT を用いた３次元偏微分方程式の解法
- is : 大規模整数ソート
- lu : Synmetric SOR iteration による CFD アプリケーション
- sp : Scalar ADI iteration による CFD アプリケーション
- bt : 5x5 block size ADI iteration による CFD アプリケーション

Y : クラス (問題のサイズ)
- A : Class A
- B : Class B
- C : Class C
- W : Workstation
- S : Sample

N : 使用ノード数

- mg.W.2 の実行例

```
:-)pload 1 2 : /ssspc/NPB2.3_bin/mg.W.2  ← pload コマンドを用いノード 1,2 へ mg.W.2 をロード
node1: argc=5 argv=[pload 1 2 : /ssspc/kanoh/tmp/mg.W.2]
load: a_magic=457f a_text=10101 a_data=0 a_bss=0 a_entry=30002
p_offset = 0x0, p_memsz = 0x3c8c2
pn = 0x3d
before memcpy(0x40000000,0x60008000,248002)
p_offset = 0x3d000, p_memsz = 0xc6df48
```

```
pn = 0xe
before memcpy(0x40000000,0x60045000,55112)
load: entry=10000074
 text addr=10000 npages=3d phypage=14b0
 data addr=1003d npages=e phypage=1740  bss addr=1004b npages=c60 phypage=ffffffff
:-)node2: pload command file_size=364326, 0=#1cbfc 1=/ssspc/kanoh/tmp/mg.W.2
load: a_magic=457f a_text=10101 a_data=0 a_bss=0 a_entry=30002
p_offset = 0x0, p_memsz = 0x3c8c2
pn = 0x3d
before memcpy(0x40000000,0x60008000,248002)
p_offset = 0x3d000, p_memsz = 0xc6df48
pn = 0xe
before memcpy(0x40000000,0x60045000,55112)
load: entry=10000074
 text addr=10000 npages=3d phypage=14d0
 data addr=1003d npages=e phypage=1510  bss addr=1004b npages=c60 phypage=ffffffff
node2: pload command finished !!

:-)pstart 1 2 : mg.W.2 -v              ← pstart コマンドを用いノード 1,2 で mg.W.2 を実行
node1: argc=6 argv=[pstart 1 2 : mg.W.2 -v]
:-)

 NAS Parallel Benchmarks 2.3 -- MG Benchmark

 No input file. Using compiled defaults
 Size:  64x 64x 64  (class W)
 Iterations:  40
 Number of processes:     2

 Initialization time:           0.299 seconds

 Benchmark completed
 VERIFICATION SUCCESSFUL
 L2 Norm is   0.119216682911E-17
 Error is    -0.131174714219E-17


 MG Benchmark Completed.
 Class           =                        W
 Size            =                64x 64x 64
 Iterations      =                       40
 Time in seconds =                     4.36
 Total processes =                        2
 Compiled procs  =                        2
 Mop/s total     =                   139.47
 Mop/s/process   =                    69.73
 Operation type  =           floating point
 Verification    =               SUCCESSFUL
 Version         =                      2.3
 Compile date    =              23 Nov 2003

 Compile options:
    MPIF77       = $(F77)
    FLINK        = $(LD)
    FMPI_LIB     = $(LIBMPIF) $(LIBMPI) $(LIBRARIES)
    FMPI_INC     = -I$(MPIDIR)/include
    FFLAGS       = $(F77FLAGS) -O3 -w
    FLINKFLAGS   = $(LDFLAGS) $(RUNTIME)
    RAND         = randdp
```

```
 Please send the results of this run to:

 NPB Development Team
 Internet: npb@nas.nasa.gov

 If email is not available, send this to:

 MS T27A-1
 NASA Ames Research Center
 Moffett Field, CA  94035-1000

 Fax: 415-604-3957



 !! Task(-V____:0x39600000) Finished !! (cmd="mg.W.2",ret=0)

 !! Task(-V____:0x39600000) Finished !! (cmd="mg.W.2",ret=0)
```

6.1.1 NAS Parallel Benchmarks のコンパイル

NAS Parallel Benchmarks(NPB) の主だったプログラムは/usr/sss/ssspc/NPB2.3_bin に用意されていますが、更に多くのノードを用いて検証する場合等、次の要領でプログラムを作成 (コンパイル) して下さい。

NAS Parallel Benchmarks(NPB) のソースコードは、次の URL から入手することができます。
MPI(Message Passing Interface) に対応した NPB2.3 を入手して下さい。

- NBP ソースコードの入手先

```
http://www.nas.nasa.gov/Software/NPB/
```

ソースコードを入手した後、適当なディレクトリ (以下の例では /tmp) に展開し、/usr/local/sss/mpi/make.def.NPB2.3 を config/make.def としてコピーして下さい。

- ソースコードの展開

```
# cd /tmp
# tar zxvf NPB2.3.tar.gz
# cp /usr/local/sss/mpi/make.def.NPB2.3 /tmp/NPB2.3/NPB2.3-MPI/config/make.def
```

/tmp/NPB2.3/NPB2.3-MPI に移り、コンパイルを実行します。　プログラムが /tmp/NPB2.3/NPB2.3-MPI/bin/ に作成されます。

- コンパイル例 (mg.W.2)

```
# cd /tmp/NPB2.3/NPB2.3-MPI/
# gmake mg NPROCS=2 CLASS=W
```

7 クロス開発環境

FreeBSD 4.8R 上で SSS-PC で動作するプログラムを作成する、クロス開発環境について説明します。

SSS-PC には、SSS-PC 専用に作られた**オリジナル libc** と FreeBSD 4.x の libc を移植した、**UNIX 互換 libc** が用意されています。　前者は SSS-PC 本来の並列処理やタスクマイグレーション機能をサポートし SSS-PC の機能と性能を十二分に活用でるものですが、UNIX の標準的な libc に対する互換がありません。　一方、後者は UNIX との互換性が高く UNIX もしくはその互換 OS に向けに作られた様々なソフトウェア (ソースコード) を活用するには便利ですが、SSS-PC 本来の機能や性能を活かし切れるものではありません。

本節では、**UNIX 互換 libc** を用いた開発環境について取り上げています。　**オリジナル libc** については、配布 CD-ROM の /src にソースコードを用意してありますので、そちらを参照して下さい。

クロス開発環境は /usr/local/sss 以下にインストールされます。　クロスコンパイラの使い方は、一般的なクロス環境と同じで、/usr/bin 以下の gcc や ld,as の代わりに /usr/local/sss/bin/i386-unknown-ssscore3.elf-* を用います。
ただし、SSS-PC は Dynamic Executable をサポートしないため、Libray は全て static にリンクする必要があります。

簡単なソース (foo.c) をコンパイルする実例を示します。

- 簡単なクロスコンパイル

```
# cat foo.c
#include <stdio.h>
#include <math.h>
{
    double x=0.5, y;
    y = cos(x);
    printf("a=%f b=%f\n",a,b);
}
# /usr/local/sss/bin/i386-unknown-ssscore3.elf-gcc -static -s -o foo foo.c -lm
```

7.1 Makefile

コンパイルしようとするソースに Makefile が用意されている場合は、Makefile において参照されている gcc や ld, as 等を書き換えます。
Makefile において`$(CC)` や `$(LD),$(AS)` 等の標準的な変数が用いられている場合は、明示的に書き換えておきます。

- Makefile に明示すべき主な変数

```
LDFLAGS+= -static -s
AR= /usr/local/sss/bin/i386-unknown-ssscore3.elf-ar
AS= /usr/local/sss/bin/i386-unknown-ssscore3.elf-as
CC= /usr/local/sss/bin/i386-unknown-ssscore3.elf-gcc
CXX= /usr/local/sss/bin/i386-unknown-ssscore3.elf-g++
FC= /usr/local/sss/bin/i386-unknown-ssscore3.elf-g77
LD= /usr/local/sss/bin/i386-unknown-ssscore3.elf-ld
NM= /usr/local/sss/bin/i386-unknown-ssscore3.elf-nm
RANLIB= /usr/local/sss/bin/i386-unknown-ssscore3.elf-ranlib
```

  - make(1) のコマンド引数に 'make CC=i386-unknown-ssscore3.elf-gcc' として渡すこともできます。

Makefile が BSD make 用に書かれたもので、かつ Makefile に上記のような変数が用いられている場合は、/usr/bin/make の代わりに /usr/local/sss/bin/i386-unknown-ssscore3.elf-make を使うことで、変数を置き換えてくれます。

7.2 configure

GNU autoconf で作成された configure が用意されている場合は、–target オプションに i386-unknown-ssscore3.elf を指定します。

- configure が用意されている場合

```
LDFLAGS="-static -s" configure \
       --target=i386-unknown-ssscore3.elf \
       --host=i386-unknown-freebsd4.8
```

  - –host=i386-unknown-freebsd4.8 は通常自動判定されるため付ける必要はありません。
  - 環境変数として LDFLAGS の他に CC や AS,LD を渡さないと旨く行かない場合があります。
  - GNU make 実行時にコマンド引数として CC や AS,LD を渡さないと旨くコンパイルできない場合もあります。

7.3 UNIX 互換 libc の互換性

SSS-PC の libc は FreeBSD 4.x から移植されたもので、高い互換性を実現していますが、一部の syscall() もしくは ioctl() と mmap() については、未実装となっています。

7.4 バイナリの確認

クロス開発においてよくある間違いとして、ターゲット (SSS-PC) 用にコンパイルしたつもりが、ホスト (FreeBSD) 用のバイナリを作成してしまう場合があります。
objedump(1) を用いて両者のバイナリを比較してみると以下に示す通り SSS-PC 用が <.text>: で始まるのに対し、FreeBSD 用は <.init>: から始まります。　SSS-PC 上へ load する前に確認してみるとよいでしょう。

- SSS-PC 用に作られたバイナリ

```
# i386-unknown-ssscore3.elf-gcc -static -s -o foo-ssscore3 foo.c -lm
# objedump -D foo-ssscore3 | more

foo-ssscore3:     file format elf32-i386

Disassembly of section .text:

10000074 <.text>:
10000074:       55                      push   %ebp
10000075:       89 e5                   mov    %esp,%ebp
10000077:       56                      push   %esi
10000078:       53                      push   %ebx
10000079:       83 ec 38                sub    $0x38,%esp
1000007c:       8d 45 d8                lea    0xffffffd8(%ebp),%eax
    <以下省略>
```

- FreeBSD 用に作られたバイナリ

```
# gcc -static -s -o foo-freebsd48 foo.c -lm
# objedump -D foo-freebsd48 | more

foo-i386-unknown-freebsd4.8:     file format elf32-i386

Disassembly of section .init:

080480ac <.init>:
 80480ac:       e8 fb 00 00 00          call   0x80481ac
 80480b1:       e8 36 9c 00 00          call   0x8051cec
 80480b6:       c3                      ret
Disassembly of section .text:

080480b8 <.text>:
 80480b8:       55                      push   %ebp
 80480b9:       89 e5                   mov    %esp,%ebp
    <以下省略>
```

7.5 デバック

NikoMon Shell において LIBCDEBUG 環境変数を設定すると libc 内に埋め込まれたデバック用のメッセージを得ることができます。

- LIBCDEBUG の設定

```
:-)putenv LIBCDEBUG="3,all"
```

  - 引数の内 '3' はデバックレベル、'all' はカテゴリを示します。
  - 設定可能なデバックレベルは、1〜3 です。
  - カテゴリについては <ssspc/sssdebug.h> を参照して下さい。
  - 引数の順序は "all,3" としても構いません。

LIBCDEBUG 環境変数に 'syslog' を指定すると、メッセージを LOGHOST 環境変数に設定したホストの syslog へ飛ばすことができます。

- メッセージを 192.168.1.100 の syslog へ飛ばす設定

```
:-)putenv LOGHOST="192.168.1.100"
:-)putenv LIBCDEBUG="syslog,3,all"
```

デバックしたいソースに次のようなコードを埋め込むことで、libc に埋め込まれたデバック用のメッセージと同じ扱いでメッセージを得ることができます。
execve() で呼び出されるオブジェクトのデバックを行う場合や、メッセージを syslog に飛ばしたい時等に便利です。

- ソースに埋め込むデバックメッセージの例

```
#include <ssspc/sssdebug.h>
any_status = any_function();
S3LDPRINTF __D((LIBC_USER, 1, "any_function()=%d\n",any_status));
```

  - 引数の内 '1' はデバックレベル、'LIBC_USER' については <ssspc/sssdebug.h> を参照して下さい。

8 NikoMon Shell コマンドリファレンス

apstart

NAME

apstart - 並列非同期タスクの実行

SYNOPSIS

apstart ⟨*pnid*⟩ [⟨*pnid*⟩] : ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

apstart ⟨*host_exp*⟩ : ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

DESCRPTION

pstart コマンドが最初にタスク群をセットアップしてから同期を取って、タスク群をスタートさせるのに対し、**apstart** コマンドはセットアップした端からタスクをスタートさせます。

apstart コマンドが利用可能なプログラムは、自ら並列タスク間の同期を取るようにコーディングされたものに限ります。

SEE ALSO

pstart

bg

NAME

bg - ジョブコントロールリストからタスクのエントリを消去する

SYNOPSIS

bg ⟨*task name*⟩

DESCRPTION

bg コマンドはョブコントロールリストからタスクのエントリを消去します。　ただし、キー入力待ちのタスクはリストから消去できません。

EXAMPLES

例えば、**jobs** コマンドの出力が次のように得られている場合、'**bg %2**', '**bg R_MA_D**' 二つのコマンドはいずれも、R_MA_D のタスクをリストから消去します。　ただし、R_MA_D がキー入力を要求すると再びリストに登録されます。

```
:-)jobs
argc=1 argv=[jobs]
[0] #4977        (sleeping by some reason)       cmd=[mon3]
[1] R_RT_D       (sleeping by some reason)       cmd=[rrtd2]
[2] R_MA_D       (sleeping by some reason)       cmd=[rmandeld]
[3] #27769       (waiting key-input event)       cmd=[keytest]
```

'**bg %3**', '**bg**' の二つのコマンドはいずれも #27769 をリストから消去しようとしますが、#27769 はキー入力待ち中なので消去されません。

SEE ALSO

fg, jobs

bye

NAME

bye - Shell をクローズする

SYNOPSIS

bye

DESCRPTION

NikoMon Shell をクローズします。　telnet によるリモート接続セッションをクローズ際に用います。

cat

NAME

cat - ファイルの連結、表示を行う

SYNOPSIS

cat [⟨*host*⟩:]⟨*file*⟩...

DESCRPTION

cat コマンドは tftp を用いて tftp サーバよりファイルを連続的に読み込み、標準出力に書き出します。

host を指定した場合、*host* に指定された tftp サーバからファイルを読み込みます。

clear

NAME

clear - システムメッセージ表示領域、グラフィック表示を消去す

SYNOPSIS

clear

DESCRPTION

コンソール画面左情報のシステムメッセージ表示領域およびグラフィック表示を消去します。

システムメッセージ表示領域を再び表示するには、Meta-P などでシステムメッセージ表示領域をスクロールさせることで再描画されます。

colors

NAME

colors - グラデーションのある縞模様を描画する

SYNOPSIS

colors

DESCRPTION

コンソール画面左方にグラデーションのある縞模様を描画します。デモ用のコマンドです。　clear コマンドで消去します。

command

NAME

command - 利用可能なコマンドを表示する

SYNOPSIS

command [**-a**]

DESCRPTION

command は利用可能なコマンドを表示します。

オプションは以下のとおりです。

-a　　組み込みコマンドを含めて表示します。

echo

NAME

echo - 引数の文字列を標準出力に出力する

SYNOPSIS

echo [*strings ...*]

DESCRPTION

echo は、1 行あたりに 1 引数の形式で標準出力に書き出します。

EXAMPLES

```
:-)echo aa bb cc
node1: argc=4 argv=[echo aa bb cc]
argv[1]=2"aa"
argv[2]=2"bb"
argv[3]=2"cc"
aa
bb
cc
:-)echo "aa bb cc"
node1: argc=2 argv=[echo aa bb cc]
argv[1]=8"aa bb cc"
aa bb cc
:-)
```

env

NAME

env - 環境変数の表示

SYNOPSIS

env

DESCRPTION

env は、環境変数の名前と値を、1 行あたりに 1 組の名前と値の形式で表示します。

EXAMPLES

```
:-)env
node1: argc=1 argv=[env]
USER=user1
HOME=/ssspc/user1
UID=1001
GID=20
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/ssspc/nikomon
LOGHOST=192.168.1.100
DISPLAY=192.168.1.100:0
TERM=nikomon
TERMCAP=console|nikomon:bl=^G:cm=5\E[%i%d;%dH:so=2\E[7m:se=2\E[m:
        ce=3\E[K:cl=50\E[H\E[J:up=\E[A:do=\E[B:nd=\E[C:le=\E[D:
```

extfs

NAME

extfs - 外部委託ファイルシステムのキャッシュ制御

SYNOPSIS

extfs dump [⟨*file*⟩

extfs kill ⟨*file*⟩ [⟨*No.*⟩]

extfs cache [{**stop**,**resume**}]

DESCRPTION

extfs は、外部委託ファイルシステムのキャッシュの制御を行うコマンドです。ノード数が数十、数百と増えた場合、外部委託ファイルシステムに対するアクセスがオーバーヘッドとなりうることから、SSS-PC ではファイルサイズが 64KB 以下のファイルをトータル 4MB までメモリ上にキャッシュします。　キャシュされたファイルはプログラムから syscall() により、もしは **extfs** の **kill** オプションを用いて明示的に削除しない限り、書き換え (または削除) られることはありません。　キャシュファイルのトータル が 4MB を越えた場合、キャシュは行われなくなります。

dump

キャシュされているファイルのステータス一覧を表示します。

dump *file*

file に指定されるファイルのステータスを表示します。

kill *file*

file に指定されるファイルをファイルキャッシュから削除します。

kill *file No.*

同一のファイルが複数のキャッシュに入る場合があります。　この場合、一つだけが有効で残りのファイルには K(to be killed) のフラグが付加されます。

'**extfs dump** *file*' で同一名エントリを dump し、そこに表示される番号 *No.* を kill します。

cache

ファイルキャッシュの動作状況 (動作中 or 停止中) を表示 s ます。

cache stop

ファイルキャッシュの機能を停止します。

cache resume

ファイルキャッシュの機能を再開します。

fg

NAME

fg - キー入力を特定のタスクに割り当てる

SYNOPSIS

fg ⟨*task name*⟩

DESCRPTION

fg コマンドは、キー入力を特定のタスクに割り当てます。

EXAMPLES

例えば、**jobs** コマンドの出力が次のように得られている場合、'**fg**', '**fg %3**', '**fg #27769**' 三つのコマンドはいずれも、#27769 のタスクにキー入力が切り替わります。　このタスクは実際にキー入力待ち状態ですので、キー入力の利を獲得してキー入力があると起こされます。

```
:-)jobs
argc=1 argv=[jobs]
[0] #4977        (sleeping by some reason)       cmd=[mon3]
[1] R_RT_D       (sleeping by some reason)       cmd=[rrtd2]
[2] R_MA_D       (sleeping by some reason)       cmd=[rmandeld]
[3] #27769       (waiting key-input event)       cmd=[keytest]
```

'**fg %1**' によって R_RT_D にキー入力の権利を渡すこともできます。　PID R_RT_D はまだキー入力を要求していませんので、キー入力は Shell 内にバッファリングされ、sss_getchar で実際に要求したときにタスクに供給されます。

なお、Shell に制御を戻す場合は、Ctrl+Z または Ctrl+C を用います.

SEE ALSO

bg, jobs

icon

NAME

icon - フォントを表示する

SYNOPSIS

icon

DESCRPTION

画面左方に各種フォントを表示します。デモ用のコマンドです。　clear コマンドで消去します。

ipconf

NAME

ipconf - IP ネットワークのパラメタ設定

SYNOPSIS

ipconf setval ⟨*netdev*⟩ ⟨*keyword1*⟩ ⟨*value*⟩

ipconf setstr ⟨*netdev*⟩ ⟨*keyword2*⟩ ⟨*string*⟩

ipconf clear ⟨*netdev*⟩ ⟨{*keyword1*,*keyword2*}⟩

ipconf query_netdev ⟨*ip_addr*⟩

keyword1: **hostaddr**, **netmask**, **defaultgateway**, **tftpaddr**, **vip0**, **vmask0**,
vip1, **vmask1**, **vipserv_type**
keyword2: **hostname**, **ipdomain**, **ypdomain**

DESCRPTION

ipconf は、IP ネットワークのパラメタを設定もしくはクリアするためのコマンドです。

ipconf setval *netdev keyword1 value*

ネットワークデバイス *netdev* の *keyword1* を *value* に指定された値 (IP アドレスもしくはネットワークアドレス) に設定します。

ipconf setstr *netdev keyword2 string*

ネットワークデバイス *netdev* の *keyword2* を *strings* に指定された文字列に設定します。

ipconf clear *netdev keyword1*

ネットワークデバイス *netdev* について、*keyword1* に設定された値 (IP アドレスもしくはネットワークアドレス) をクリアします。

ipconf clear *netdev keyword2*

ネットワークデバイス *netdev* について、*keyword2* に設定された文字列をクリアします。

ipconf query_netdev *ip_addr*

netdev

netdev は、**ipstat** コマンドで得られる利用可能なネットワークデバイスの番号です。

value

192.168.1.100 や 255.255.255.0 のように、インターネット標準のドット表記で表される IP アドレスもしくはネットワークアドレスです。

keyword1

hostaddr

ホストの IP アドレス

netmask

ネットマスク

defaultgateway

デフォルトゲートウェイ

tftpaddr

TFTP サーバのアドレス

vip0,**vip1**

仮想 IP アドレス

vmask0,**vmask1**

仮想 IP アドレスのネットマスク

vipserv_type

keyword2

hostname

ホスト名

ipdomain

IP ドメイン名

ypdomain

YP ドメイン名

EXAMPLES

以下に設定例を示します。

```
ipconf setval 0 netmask 255.255.255.0
ipconf setval 0 defaultgateway 192.168.1.1
ipconf setstr 0 hostname "sss-pc1"
ipconf setstr 0 ipdomain "example.jp"
ipconf setstr 0 ypdomain "YP-Example-Domain"
```

ipstat

NAME

ipstat - IP ネットワークの設定状況を表示します。

SYNOPSIS

ipstat

DESCRPTION

ipstat コマンドは、IP ネットワークの設定状況を表示します。

EXAMPLES

以下に実行例を示します。

```
:-)ipstat
node1: argc=1 argv=[ipstat]
netdev=0: NIC type="eepro100"
hostaddr=192.168.1.101, netmask=255.255.255.0
defaultgateway=192.168.1.1
hostname="sss-pc1"
ipdomain="example.jp"
ypdomain="YP-Example-Domain"

tftpaddr (for nikomon load) = 192.168.1.100
:-)
```

jobs

NAME

jobs - 過去に起動して、終了していないタスクのリスト表示

SYNOPSIS

jobs

DESCRPTION

jobs コマンドは、過去に起動して、終了していないタスクのリストを表示します。　新しいタスクはリストの最後に追加されます。

EXAMPLES 以下に実例を示します。

```
:-)jobs
argc=1 argv=[jobs]
[0] #4977        (sleeping by some reason)       cmd=[mon3]
[1] R_RT_D       (sleeping by some reason)       cmd=[rrtd2]
[2] R_MA_D       (sleeping by some reason)       cmd=[rmandeld]
[3] #27769       (waiting key-input event)       cmd=[keytest]
```

bg, **fg**, **kill** 等、タスク名を指定するコマンドにおいては、タスク名の代わりにタスクリストに示される番号 (**%0** や **%1**) を用いることができます。　**%%** **は最新のタスク**を示します。

例えば、上記実行例において '**kill %1**' では R_RT_D が、'**kill %%**' では #27769 がそれぞれ終了させられます。

SEE ALSO

bg, fg, kill

kill

NAME

kill - タスクの強制終了、プログラムをメモリから削除する

SYNOPSIS

kill [**-s**] ⟨*task name*⟩

kill ⟨*file*⟩

DESCRPTION

kill コマンドは、動作しているタスクの強制終了もしくは **load** コマンド等によりメモリにロードされたプログラムの削除に用いられます。

-s コンソールから他の shell で動作しているタスクを終了する場合に用います。

SEE ALSO

jobs, ps, command

lcd

NAME

lcd - tftp サーバの current パスを設定する

SYNOPSIS

lcd ⟨*path*⟩

DESCRPTION

lcd コマンドは **cat** コマンドや **load** コマンド等が tftp サーバにアクセスする際のデフォルトパス ($cwd 環境変数) を設定します。

EXAMPLES

次の実行例は等価です。

```
lcd /sspc
putenv cwd="/sspc"
```

SEE ALSO

pwd, env, putenv

load

NAME

load - プログラムをメモリへロードする

SYNOPSIS

load [⟨*host*⟩:]⟨*file*⟩ [⟨*command string*⟩]

DESCRPTION

load コマンドは tftp を用いて tftp サーバよりプログラムをメモリへ読み込みます。
host を指定した場合、*host* に指定された tftp サーバからプログラムをメモリへ読み込みます。

lstftp

NAME

lstftp - tftp サーバのファイル一覧を取得する

SYNOPSIS

lstftp ⟨*path*⟩

DESCRPTION

lstftp コマンドは tftp サーバのファイル一覧を取得するコマンドです。
lstftp コマンドを利用するためには tftp サーバで **freebsd_fs** が動作し、かつ /usr/sss/freebsd/boot 下に FreeBSD の ls(1) と sh(1), rm(1) が用意されている必要があります。

EXAMPLES

```
:-)lstftp /usr
node1: argc=2 argv=[lstftp /usr]
:-)FreeBSD command:/freebsd/boot/ls //usr
bin
i386-unknown-ssscore3.elf
include
info
lib
libexec
local
man
sbin
share
```

msh

NAME

msh - Remote Shell for MBCF

SYNOPSIS

msh ⟨*pnid*⟩ ⟨*command*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ; ...]
msh ⟨*pnid*⟩ ⟨*command string*⟩ [; ⟨*command string*⟩ ; ...]

DESCRPTION

msh は rsh 機能を MBCF(Memory-Based Communication Facilities) でセキュア (shell はカーネルレベルであるため、ユーザレベルタスクを介して MBCF 通信を行っている) に実現したものです。
command もしくは *command string* に指定したコマンドを *pnid* に指定された遠隔ノードで実行します。
ただし、rsh とは異なり **msh** コマンド実行時実行場所の環境を遠隔ノードに持ち込んで、コマンドを実行します。

pnid

pnid(Parallel Node ID) 遠隔ノードのノード番号

command string

説明上 '*command arg1 arg2 ...*' を意図したもの、通常のコマンド行です。
';' で *command string* を区切り複数のコマンドも連続起動することも可能です。　連続起動時において、起動はシーケンシャルに行われますが、先行するコマンドの終了を待たずに次のコマンドが実行されます。　一般的な **sh** に実装されている先行するコマンドの終了を待って次のコマンドを実行する機能、いわゆる && 的な機能は、現在のところ NikoMon Shell には実装されていません。

SEE ALSO

psh

myname

NAME

myname - NikoMon Shell のタスク名を表示

SYNOPSIS

myname

DESCRPTION

myname コマンドは動作している NikoMon Shell のタスク名を表示します。

objload

NAME

objload - コード名を指定してプログラムをメモリにロードします。

SYNOPSIS

objload ⟨*code name*⟩ [⟨*host*⟩:]⟨*file*⟩ [⟨*command string*⟩]

DESCRPTION

objload コマンドはコード名 (SSS-PC の 6 文字名前) を指定してプログラムをメモリにロードします。コード名を指定する点を除き、**load** コマンドと同じです。　コード名はプログラムが実行される際の、タスク名となります。

SEE ALSO

load, oblist

oblist

NAME

oblist - メモリ上のオブジェクト一覧を表示する

SYNOPSIS

oblist [**-a**]

DESCRPTION

oblist コマンドはメモリ上のオブジェクトの一覧を表示します。
オプションは以下のとおりです。
-a　　ノードの統べてのオブジェクトについて表示します。

各フィールドは左からコード名 (SSS-PC の 6 文字名前)、オブェクトの種別、アロケーション、クリエータのタスク名です。

pload

NAME

pload - Parallel Load

SYNOPSIS

pload ⟨*pnid1*⟩ [⟨*pnid2*⟩ ...] : *file*

pload ⟨*pnid1 – pnid2*⟩ [⟨*host_exp*⟩ ...] : *file*

DESCRPTION

pload コマンドは **load** コマンドの並列処理版です。

file に指定されたプログラムを *pnid1*, *pnid2* または '*pnid1 – pnid2*' もしくは *host_exp* に指定された遠隔ノードへロードします。

file に指定されたプログラムは tftp を用いて tftp サーバより、**pload** コマンドを実行したノードへ読み込まれ、MBCF(Memory-Based Communication Facilities) を介して *pnid1*, *pnid2* または '*pnid1 – pnid2*' もしくは *host_exp* に指定される遠隔ノードへコピーされます。

pload コマンドは **msh** コマンド同様、**pload** コマンド実行時実行場所の環境を遠隔ノードへ引き継ぎます。　このためディレクトリの移動 (**lcd** コマンド) 等はコマンド実行場所でのみ行えば済みます。

EXAMPLES

以下に示す **pload** コマンドと **psh** コマンドの実行例は、機能的に等価ですが、**pload** は最初の tftp による load 以外は MBCF でプログラムをコピーするため高速かつ安全です。(tftp は複数重なると失敗する可能性があります。)

```
pload 1 -- 4 : command_file
psh 1 -- 4 : load command_file
```

SEE ALSO

load, msh, psh

pobjload

NAME

pobjload - コード名を指定した Parallel Load

SYNOPSIS

pobjload ⟨*code name*⟩ ⟨*pnid1*⟩ [⟨*pnid2*⟩ ...] : *file*

pobjload ⟨*code name*⟩ ⟨*pnid1 – pnid2*⟩ [⟨*host_exp*⟩ ...] : *file*

DESCRPTION

pobjload コマンドはコード名 (SSS-CORE の 6 文字名前) を指定してプログラムをメモリにロードします。

コード名を指定する点を除き、**pload** コマンドと同じです。　コード名はプログラムが実行される際の、タスク名となります。

SEE ALSO

pload, oblist

ps

NAME

ps - タスクの状態を表示

SYNOPSIS

ps [**-ac**]

DESCRPTION

ps コマンドはタスクの状態を表示します。オプションは以下のとおりです。

-a ノードの統べてのタスクについて表示します。

-c pri, sli, life, s.life に代えて comments を表示します。

各フィールドは次のとおりです。

name

タスク名

creator

タスクを起動させたタスク

type

タスクの種別 p_sys, np_sys, p_task

stat

ready, sleep, wait, active

w.typ

待機種別 none, queR, evnt

w.obj

待機しているタスク

tout

pri

sli

life

s.life

comments

psh

NAME

psh - Parallel remote Shell for MBCF

SYNOPSIS

psh ⟨*pnid1*⟩ [⟨*pnid2*⟩ ...] : ⟨*command*⟩ [⟨*arg1*⟩ ...]

psh ⟨*pnid1 – pnid2*⟩ [⟨*host_exp*⟩ ...] : ⟨*command*⟩ [⟨*arg1*⟩ ...]

DESCRPTION

psh コマンドは **msh** コマンドの並列処理版です。 *command* に指定されたコマンドを *pnid1*, *pnid2* または *'pnid1 – pnid2'* もしくは *host_exp* に指定れた遠隔ノードで実行します。

pnid1 – pnid2

'pnid1 ≤ *pnid2* で連続した pnid を補間する shell の parse 機能です。 '–' の前後に必ず SPACE か TAB が必要となります。

host_exp

'–' や個別 pnid の並びを利用して command を並列実行させたいノードのリストを意図します。　環境変数 (*$⟨valname⟩*) も使用可能です。

psh は **msh** 同様、**psh** コマンド実行時実行場所の環境が遠隔ノードでも引き継がれます。

EXAMPLES

次の実行例は遠隔ノード 1,2,3,4,6,8 を再起動するものです。`psh 1 -- 4 6 8 : reboot`

SEE ALSO

msh

pstart

NAME

pstart - 並列タスクの実行

SYNOPSIS

pstart ⟨*pnid*⟩ [⟨*pnid*⟩] : ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

pstart ⟨*host_exp*⟩ : ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

DESCRPTION

pstart コマンドは **start** コマンドの並列処理版です。　*command* に指定されたコマンドに *task name* で指定されたタスク名を与え、*pnid1*, *pnid2* または *host_exp* に指定された遠隔ノードで実行します。　**psh** コマンドを用いて

psh ⟨*host_exp*⟩ : start ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

とした場合との違いは、MBCF のための初期設定が行われる点です。
host_exp が '1 3 5 2 6 8' である場合、論理タスクの 0 が (*pnid*=1,⟨*task name*⟩)、1 が (*pnid*=3,⟨*task name*⟩)、2 が (*pnid*=5,⟨*task name*⟩)、3 が (*pnid*=2,⟨*task name*⟩)、4 が (*pnid*=6,⟨*task name*⟩)、5 が (*pnid*=8,⟨*task name*⟩) というように初期化され、accesskey として自動生成された同一の値が設定されます。

また、pstart は同一ノードで複数の同一コマンドによるタスクを起動することができます。

```
pstart 1 2 3 3 3 4 2 : pstest pst arg1 arg2
```

以上のように実行すると (1,pst_),(2,pst_),(3,pst_),(3,psta),(3,pstb),(4,pst_),(2,psta) (SSS–PC の 6 文字名前では大文字小文字を区別せず '_' は NIL と等価です。) の 7 個のタスク (論理番号は 0 から順番に 6 まで) が生成されます。このように複数タスクを同一ノードで起動する場合はタスク名を５文字以下で指定して下さい。なお、同一ノードで起動できるパラレルタスクの最大数は 32 です。
pstart コマンドは最初にタスク群をセットアップしてから同期を取って、タスク群をスタートさせます。　このためプログラム中ですぐに MBCF 通信を行えます。　これに対して **apstart** コマンドはセットアップした端からタスクをスタートさせます。　自分で最初の同期をとる場合は **apstart** コマンドも使用可能です。

SEE ALSO

psh, apstart, start

putenv

NAME

putenv - 環境変数に値を代入する。

SYNOPSIS

putenv ⟨*envstr*⟩

DESCRPTION

putenv コマンドは環境変数に値を代入します。

envstr

envstr は、' 環境変数名=値' の形式で指定します。　' 値' には、環境変数に応じ数字、文字、文字列が用いられます。

NikoMon には unsetenv に相当する機能が用意されていないため、環境変数名をクリアした場合は、NULL(“”') を代入して下さい。

pwd

NAME

pwd - tftp サーバの current パスを表示する。

SYNOPSIS

pwd

DESCRPTION

pwd コマンドは **lcd** コマンドもしくは **putenv** コマンドにより設定された tftp サーバにアクセスする際のデフォルトパス ($cwd 環境変数) を表示します。

EXAMPLES

次の実行例は等価です。

```
pwd
echo $cwd
```

SEE ALSO

pwd, env, putenv

reboot

NAME

reboot - ノード (マシン) を再起動する。

SYNOPSIS

reboot

DESCRPTION

reboot コマンドはノード (マシン) を再起動します。

resume

NAME

resume - stop コマンドで中断させたタスクを再開する

SYNOPSIS

resume ⟨*task name*⟩

DESCRPTION

resume コマンドは **stop** コマンドで中断させたタスクを再開します。　*task name* は再開させるタスクのタスク名です。

SEE ALSO

resume, taskque, regdump, stkdump, memdump, taskdump

signal

NAME

signal - Signal の操作

SYNOPSIS

signal [**-sv**] **-send** ⟨*task name*⟩ {⟨sig_vector⟩,⟨sig_value⟩}

signal [**-v**] **-query** ⟨*task name*⟩

DESCRPTION

signal コマンドは *task name* に指定されるタスクに *sig_vector* または *sig_value* を送ります。

-send

signal を送ります。

-query

UNIX タイプ signal 関連の状態が表示が表示されます。

オプションは以下のとおりです。

-s　コンソールから他の shell へ signal を送る場合に用います。

-v　イベント名を表示します。

EXAMPLES

```
signal -send vrt -HUP
signal -send vrt SIGFPE
signal -send vrt 0x10002
signal -send %1 0x10002
```

start

NAME

start - タスク名を明示的に指定してプログラムを実行する

SYNOPSIS

start ⟨*command*⟩ ⟨*task name*⟩ [⟨*arg1*⟩ ⟨*arg2*⟩ ...]

DESCRPTION

start コマンドは、*command* に指定されたコマンドに *task name* で指定されたタスク名を与えて実行します。

EXAMPLES

以下の例は、'lynx' というプログラムに 'LYNX' というタスク名を与え実行するものです。　プログラム 'lynx' の引数には URL が与えられています。

```
start lynx LYNX lynx "http://192.168.1.100/"
```

stkdump

NAME

stkdump - タスクのスタック情報をダンプする

SYNOPSIS

stkdump ⟨*task name*⟩ [*level* [*size*]]

DESCRPTION

stkdump コマンドはタスクのスタック情報をダンプします。

SEE ALSO

stop, resume, taskque, regdump, memdump, taskdump

stop

NAME

stop - プログラムの実行を中断する

SYNOPSIS

stop ⟨*task name*⟩

DESCRPTION

stop コマンドは *task name* に指定したタスク名のタスクを中断させます。

stop コマンドは、**taskque**, **regdump**, **stkdump**, **memdump**, **taskdump** 等のタスク情報を得るコマンドを用いる際に使用します。

SEE ALSO

resume, taskque, regdump, stkdump, memdump, taskdump

stty

NAME

stty - 端末の設定を行なう

SYNOPSIS

stty {**-a**,**all**,⟨*option*⟩ ⟨*value*⟩}

DESCRPTION

stty コマンドは端末の設定／表示を行うコマンドです。

-a,**all**

端末の設定状態を表示します。

option value

端末の属性を設定します。

submit

NAME

submit - submit ファイルをロードする

SYNOPSIS

submit [⟨*host*⟩:]⟨*file*⟩

DESCRPTION

submit コマンドは tftp を用いて tftp サーバより *file* に指定した submit ファイルを読み込み、submit ファイルに記載されたコマンドを記載された順に実行します。
host を指定した場合、*host* に指定された tftp サーバから submit ファイルを読み込みます。

開発元

株式会社情報科学研究所	http://www.isll.co.jp/
SSS-PC プロジェクトチーム	http://www.ssspc.org/
国立情報学研究所　松本　尚	http://www.nii.ac.jp/
お問い合わせ先	info@isll.co.jp

本ソフトは、IPA (情報処理振興事業協会) の 『2003 年度重点領域情報技術開発事業』の支援をうけて開発されたものです。

以上