実践で学ぶ

# 一歩進んだサーバー構築・運用術

written by 仙石 浩明

## 最終回 initとgetty

**連載最終回の今回は，initとgettyを取り上げます。initはLinuxを搭載したシステムにおいて一番最初に実行されるプログラムです。gettyはネットワーク以外のデバイス(つまりモデムが接続されたシリアル・ポートや，コンソール)上のサービスを提供するプログラムです。**

初もうで帰りに電気屋をふらふらしていたら，17インチ液晶ディスプレイを衝動買いしてしまいました。もともと，7年ほど使っていたナナオ製17インチCRTの調子が悪くなってきたので液晶ディスプレイの購入を検討中だったのですが，17インチの液晶は30万円以上する*1ものと思い込んでいて，とても手が出せないから15インチで我慢しようと思っていたのが，13万5000円*2（しかも10%ポイント還元付き）のエイサー製17インチ液晶ディスプレイを店頭で見かけて，そのまま購入してしまった*3というわけです。

大きめの電気屋だと，必ずといっていいほど各メーカーのディスプレイを並べて展示していて，画質を良く見比べてお選び下さい，と言わんばかりなのですが，大抵は同じPCのビデオ信号を（安物の）分配機で分配しているので，信号が劣化しまくりで表示画像がボケボケとなり，困ったものです。ひどい店になると，解像度が異なる液晶ディスプレイに同じ信号（つまり低解像度用）を送っていたりします。

17インチだと解像度は普通，1280×1024ドットですが，このくらいの解像度になるとディスプレイの性能を見るには相当高性能の分配機*4が必要になるはずで，そういうことなら裏にPCの展示機を並べてそこから1台づつ信号を取ったほうが良いのではないかと思う今日このごろです。そもそもデモ用の画像は，表示パターンの切り替えが速すぎて，じっくり見比べることができないので好きではありません。

私が立ち寄った電気屋は，たまたまデモ用の画像を出力していたPCが故障していました。店員さんに，画質を確認したいと言うと，わざわざノートPCを持ってきてくれた*5ので，それをディスプレイに直結してじっくり観察することができました。

最初，17インチにしては破格なので，安かろう悪かろうだと思っていたのですが，実際にPCを直結して見てみると，どうしてどうして悪くないコントラスト*6です。会社で使っているナナオ製に勝るとも劣らない鮮明さで，これなら大丈夫と購入に踏み切りました。今，この原稿をその液晶ディスプレイを見ながら書いていますが，全く申し分ない性能です。

### init

さて，みなさんはLinuxシステムがブートするとき，どのような手順を踏むかを知っていますか？。DOSの場合なら，「config.sysの内容に基づき各種デバイス・ドライバが組み込まれた後，command.comが実行され，command.comはautoexec.batに記述されたコマンドを実行した後，プロンプトを表示してコマンド入力待ちになる」と，ほとんどの人が知っていることでしょう。この程度のことは，初心者向けの入門書にさえ書いてあります。なんらかのトラブルでブ

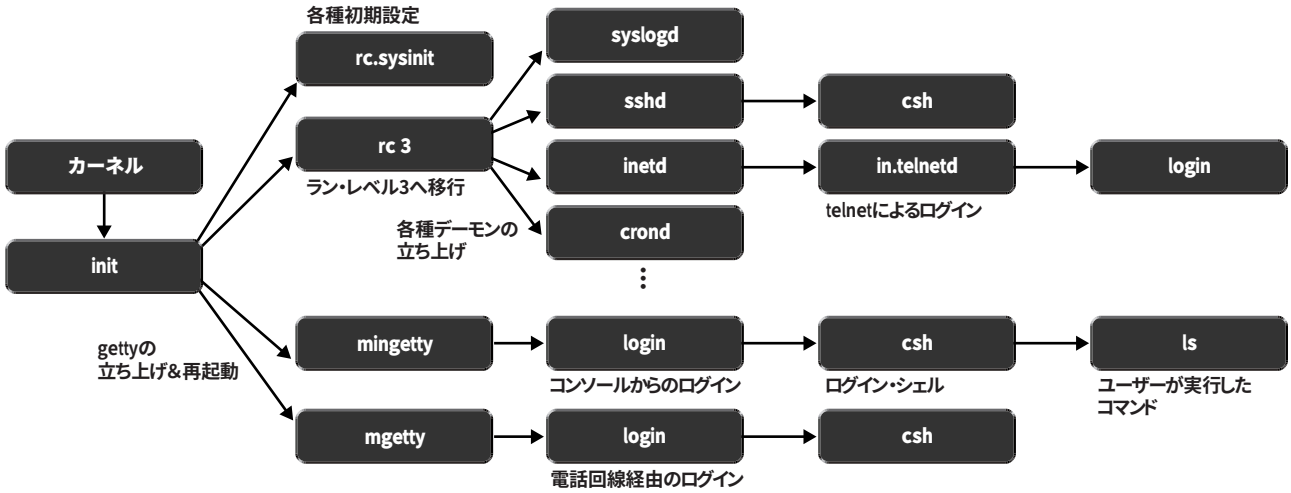


**図1　initがその他のすべてのプログラムを直接あるいは間接的に起動する**

ートしなくなったとしても，config.sysやautoexec.batを丹念に見ていけば，すぐに原因が分かることでしょう。

ところがWindowsの場合になると途端にお手上げです。ブート手順が入門書に書いてないのはもちろんのこと，上級管理者向けの専門書にすら，その全容は明らかにされていません。大多数の管理者にとってWindowsはブラック・ボックスであり，トラブル発生時の解決策は対症療法的で，ブートしなくなれば最後の手段はすべてを消してインストールし直す，ということになってしまいます。

そしてLinuxも同じ道を歩み始めています。ブート時に実行される各種設定用のスクリプトは複雑多岐にわたり，そのすべてを読みこなすことは容易ではありません。その一方で通常の管理はGUI管理ツールでお手軽にできてしまうので，ツールの裏側で何が行われているのかきちんと把握すること無く，なんとなく管理した気になってしまっている管理者が多いのではないでしょうか？。

Windowsの場合と同じく，うまく行っているときはこれでも良いのですが，うまく行かなくなる可能性のあることは必ずうまくいかなくなる＊7の言葉通り，何かのはずみ＊8にブートしなくなるなどのトラブルに見舞われることはよくあることです。そんなとき，UNIXの基礎を知っているか否かが，トラブルに対処できるか再インストールの憂き目に会うかの分かれ道になります。

というわけで，LinuxなどのUNIX互換OSのブート手順のかぎであるinitプログラムから始めましょう。

カーネルが立ち上がった後，最初に実行される＊9のが，initプログラムです。最初のプロセスですからプロセスIDは常に1になります。**図1**に示すように，その他のすべてのプログラムは，initから直接起動されるか，あるいはinitが起動したプログラムなどから間接的に起動されます。あるプログラムが他のプログラムを起動したとき，起動した側を親プロセス，起動された側を子プロセスと呼びますが，この伝でいけばinitはすべてのプロセスの共通のご先祖様＊10ということになります。

initがどのプログラムを直接起動するかは，/etc/inittabファイルで指定します。ここからはディストリビューションによって異なります。例えば私のサーバー・マシン＊11の場合は，**図2**に示す行が含まれているので，initは一番最初

＊1　前職，現職ともに会社ではナナオ製の17インチ液晶ディスプレイを買ってもらっていました。それは40万円もする，とても個人では手が届かない製品です。
＊2　15インチでも高いものは13万円以上しますから，安くなったものです。製品情報は，http://www.acer.co.jp/products/monitore/fp751.htmlを参照。
＊3　CRTだと車でない限り持ち帰り不可能なので，液晶だから徒歩でも持ち帰りできるのは良いのですが，予定外の大荷物で大変でした。
＊4　アナログ信号の場合。
＊5　安売りで有名な量販店なのですが，私1人のために店員さんが長時間つきっきりでした。安いだけでなくサービスもすばらしい店です。
＊6　私はデザイナではないので発色には興味ありません。ナナオ製に勝るとも劣らないと書きましたが，もしかすると色再現性あるいはその他の性能で，ナナオ製も高いなりの理由があるのかも知れません。
＊7　マーフィーの法則。
＊8　インストールしたパッケージの不具合(バグ)，あるいはパッケージ相互の干渉などの原因によって，パッケージ作成者の想定外の状態になってしまい，トラブルの元になることが有り得ます。
＊9　Linuxカーネルのソースのlinux/init/main.cに，initプログラムを起動するコードがあります。
＊10　ご先祖様といってもこのinitプロセスはシステムをシャットダウンするまで，つまり，どの子プロセスよりも長く生き続けます。
＊11　1994年のSlackwareをベースに，オリジナルの痕跡を全くとどめないほど手を加えたサーバー・マシンです。

```
si::sysinit:/etc/rc.d/rc.S
```

**図2　筆者のサーバー・マシンの/etc/inittabから抜粋**

**表1　ラン・レベル**

| レベル | 内容 |
|---|---|
| S | シングル・ユーザー・モード |
| 0 | システムを停止するためのモード |
| 1 | シングル・ユーザー・モードへ移行するためのモード |
| 2～5 | 管理者が自由に定義できるモード |
| 6 | システムをリブートするためのモード |

**表2　プログラム起動方法**

| 起動方法 | 内容 |
|---|---|
| sysinit | 一番最初に起動 |
| once | 指定したラン・レベルへ移行したとき実行 |
| wait | 指定したラン・レベルへ移行したとき実行し，実行終了まで待つ |
| respawn | 実行が終了したら，同じプログラムを再起動する |
| ctrlaltdel | Control + Alt + Del が押されたとき実行 |
| initdefault | デフォルトのラン・レベル |

```
si::sysinit:/etc/rc.d/rc.sysinit
```

**図3　Red Hat系ディストリビューションの/etc/inittabから抜粋**

```
id:3:initdefault:
```

**図4　ブート後にラン・レベル3へ移行する設定**

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

**図5　Red Hat系ディストリビューションのスクリプト起動（inittabから抜粋）**

```
su:1S:wait:/etc/rc.d/rc.K
rc:2345:wait:/etc/rc.d/rc.M
```

**図6　私のマシンのスクリプト起動（inittabから抜粋）**

```
c1:123:respawn:/sbin/agetty 38400 tty1
c2:123:respawn:/sbin/agetty 38400 tty2
c3:23:respawn:/sbin/agetty 38400 tty3
c4:23:respawn:/sbin/agetty 38400 tty4
c5:23:respawn:/sbin/agetty 38400 tty5
c6:2345:respawn:/sbin/agetty 38400 tty6
s0:35:respawn:/usr/local/sbin/mgetty ttyS0
s1:35:respawn:/usr/local/sbin/mgetty ttyS1
x1:45:wait:/etc/rc.d/rc.X11R6
```

**図7　ラン・レベルごとに実行するプログラムの設定（inittabから抜粋）**

```
telinit 5
```

**図8　telinitコマンドを使ってラン・レベル5に切り替える**

（sysinit，後述）に/etc/rc.d/rc.Sプログラムを実行します。Red Hat系のディストリビューションだと，**図3**に示す行が含まれているので，/etc/rc.d/rc.sysinitプログラムを実行します。rc.Sもrc.sysinitもshスクリプトで，ブート時に実行すべきプログラムが記述されています。

inittabファイルは，この例のように各行が「:」で区切られた4つのフィールドから構成されます。1番目のフィールドはIDです。1～4文字の文字列で，他の行と互いに異なる文字列でなければなりません。2番目のフィールドは，この行が適用されるラン・レベルです。ラン・レベルというのは，initが保持している動作モードのことで，一覧を**表1**に示します。3番目のフィールドは，プログラムの起動方法の指定です。主要なものを**表2**に示します。ただし，initdefaultはプログラムの起動とは関係なく，ブート時のデフォルトのラン・レベルを指定するためのものです。例えば，**図4**のように書いておけば，ブート後にラン・レベル3へ移行します。4番目のフィールドは，実行するプログラムです。

Red Hat系のディストリビューションだと，**図5**のようにラン・レベルごとに最初に実行すべきプログラムが設定されています。つまり，ラン・レベル***n***（***n***=0，1，…，6）へ移行したとき，まず「/etc/rc.d/rc ***n***」が実行されます。ちなみに私のサーバーでは，**図6**のように書いてあります。

/etc/rc.d/rcもshスクリプトで，引数に指定されたラン・レベルに移行したときに実行すべきプログラムや，そのラン・レベルで必要なデーモン類の立ち上げが行われます。ローカルな設定を記述するための/etc/rc.d/rc.localスクリプトも，この/etc/rc.d/rcから実行されます。デフォルトのラン・レベルは3ですから，ブート時は/etc/rc.d/rc 3が実行されます。つまり，前述した/etc/rc.d/rc.sysinitと，/etc/rc.d/rcのスクリプトを読めば，ブート手順のすべてが分かる，というわけです。

残念ながら最近のディストリビューションのブート・スクリプトは極めて複雑で読みこなすのは少々骨かもしれませんが，いくら複雑でもしょせんはshスクリプト[*12]ですから，さほど予備知識は必要ありません。ぜひ挑戦してみて下さい。

複雑なのは，GUI管理ツールで設定

できるような仕掛けが入っているためと，さまざまなユーザーのニーズに対応するため，過剰に汎用的であるためです。管理ツール任せ＊13でなく手作業で必要十分なスクリプトを書けば，ずっと小さくなるでしょう。私のマシンだと，ブート・スクリプトから呼び出されるスクリプトを合計しても全部で百数十行程度で済んでいます。

さて，ラン・レベル2〜5は管理者が自由に定義できる＊14ので，後述するgettyやX Window Systemの実行の有無をラン・レベルごとに変えると便利です。例えば私は，**図7**のように設定しています。この場合，ラン・レベルによってコンソールを監視するagettyの数を変えています。ラン・レベル2あるいは3の場合，tty1〜tty6（コンソール）でagettyを実行していますが，ラン・レベル4あるいは5の場合はtty6のみです。

また，ラン・レベル3あるいは5の時，シリアル・ポートのttyS0とttyS1を監視するmgettyを実行し，ラン・レベル4あるいは5の時，X Window Systemを立ち上げてxdmを実行します。どちらも「respawn」が指定してありますから，プログラムが終了した場合は再立ち上げが行われます＊15。

ラン・レベルの切り替えは，telinitコマンドを用います。例えばデフォルト状態のラン・レベル3からラン・レベル5に切り替えるときは，root権限で**図8**のように実行します。

すると図7の最後の行のプログラムである/etc/rc.d/rc.X11R6が実行され，その結果，X Window Systemが立ち上がります。/usr/local/sbin/mgettyはラン・レベル3および5なので，ラン・レベルを3から5へ変更した場合は新たに実行されることはありません。

一方，/usr/local/sbin/mgettyの行にはラン・レベル2は含まれていませんから，ラン・レベルを3から2へ変更すると，実行中のmgettyプログラムにTERMシグナルが送られます＊16。ちなみに，現在のラン・レベルを知るには，runlevelコマンドを用います。

以上をまとめると，Linux上のプロセスは次の2種類に大別できます。

（1）initが実行するブート・スクリプトによって起動されるデーモン類，およびこのデーモン類から実行されるプロセス。

（2）initが繰り返し実行（respawn）するgetty類，およびこのgetty類から起動されるプロセス。

実を言えば，getty類を繰り返し実行するデーモンを作れば，(2)を(1)に含めることも可能なのですが，シリアル・ポートあるいはコンソールなどのキャラクタ・デバイス上のサービスを提供するgettyは，initから直接起動する＊17ことが多いようです。ネットワーク上のサービスがすべてデーモンで実現されているのとは対照的です。

## getty

gettyは，get ttyの意味で，tty＊18からのログインを受け付ける際に，接続のための各種設定を行うためのプログラムです。ネットワークに直接関係ないためか，最近はあまり取り上げられる機会がありませんが，インターネット全盛の今日でも，たまにはネットワーク経由ではなく電話線経由でログインすることもあるでしょうし＊19，トラブル時にはサーバーのコンソールを直接たたく羽目に陥る＊20こともあるでしょう。まだまだ有用な技術と言えます。

gettyは実行されると，まずttyをオープンし，ボー・レートやフロー・コントロールなどを設定した上で，ログイン・プロンプトを出力します。そして，ユーザーからIDの入力があると，/bin/loginなどを呼び出します。そしてユーザーがログアウトすると終了し，initによって再度起動されます。

Linuxではagetty（Alternative GETTY）が標準的に使われていますが，コンソール専用に機能を最小化したmingetty（MINimal GETTY）や，モデムが接続されたシリアル・ポート用に機能を

＊12　分からないプログラムが出てきたら，manを実行すれば済みますね。

13　管理者たるもの，サーバーでどんなプログラムが起動されているのか完全に把握しているべきですから，何が実行されるか把握し切れない管理ツールなど使うべきではないでしょう。ディストリビューションのブート・スクリプトを読みこなして全容が把握できたら，次はぜひ自分で書き直してみてください。起動するデーモンの数が少なければ極めて簡潔に記述できるはずです。

＊14　ただし，Red Hat系のディストリビューションの場合，未定義なのはラン・レベル4だけです。

＊15　プログラムのミスなどの原因で，立ち上げたプログラムが立ち上げ直後に終了してしまう場合，再立ち上げを永久に繰り返すとCPUの無駄です。そのため，2分間に10回以上再立ち上げが行なわれたプログラムについては，5分間再立ち上げを抑制します。

＊16　5秒待ってプロセスが終了しないときは，さらにKILLシグナルが送られます。

＊17　もっとも，gettyを起動するデーモンを作ってしまったら，initの仕事が無くなってしまいますね。

＊18　Tele TYpewriterの略。何とも古めかしい語感の言葉ですが，タイプライタとは似ても似つかない姿(と言っても，キーボード配列は昔ながらのQWERTY配列だったりしますが)になってしまった今も，コンピュータの端末装置はttyと呼ばれます。

＊19　近くにプロバイダのアクセス・ポイントが無い，あるいはアクセス・ポイントの電話番号を知らない場合，長距離通話でなければ目的のサーバーに直接電話をかけてしまった方が簡単です。また，ファイアウオールやネットワーク機器のトラブルの可能性を考えると，ネットワーク経由以外のログイン手段を用意しておくべきでしょう。

＊20　カーネルのバージョンアップ時などは，コンソールから操作したほうが不意のトラブルに対処しやすくなりますね。

充実させたmgetty(smart*21Modem GETTY)も良く使われています。

### コンソール

X Window System などを立ち上げていない状態のLinuxマシンのディスプレイおよびキーボードが,「コンソール」です。Linuxの場合, 12個の仮想コンソールを持っていて, ***n***番 (***n*** = 1, 2, ..., 12) の仮想コンソールへ切り替えるには, 左側のAltキーを押しながらファンクション・キー***n***番を押すことによって切り替えることができます。

コンソールへの入出力は, デバイス・スペシャル・ファイルの/dev/tty***n***を介して行います。例えば仮想コンソール8番に文字を出力するには, **図9**などのように実行します。普通は, /dev/tty8はroot以外の書き込みが禁止されているので, root権限で図9のコマンドを実行してください。実行後, Alt+F8を押して仮想コンソール8番へ切り替えて, 出力された文字列を確認しましょう。

仮想コンソール8番から文字を入力するには, **図10**などのように実行しておいて, Alt+F8を押して仮想コンソール8番へ切り替えて, 適当な文字列を入力しリターン・キーを押します。図10のコマンドを実行中のコンソールへ戻って, catコマンドの出力を確認してください。

以上のように, Linuxの仮想コンソールは簡単に使えますから, gettyを動かすのも簡単です。試しに仮想コンソール8番でgettyを動かしてみましょう。まず, /etc/inittabに**図11**に示す行を追加します。

ラン・レベルのフィールドは, 現在のラン・レベル(この場合は3)を設定してください。次にkillコマンドを使って, initプロセス(プロセスIDは1)にHUPシグナルを送る(**図12**)か, あるいはtelinitコマンドを使って(**図13**), initに変更後の/etc/inittabを読み込ませます。仮想コンソール8番へ切り替えて, ログイン・プロンプトが表示されているかを確認してください。

```
echo abc > /dev/tty8
```

**図9　仮想コンソール8番に文字を出力**

```
cat < /dev/tty8
```

**図10　仮想コンソール8番から文字を入力**

```
8:3:respawn:/sbin/mingetty tty8
```

**図11　仮想コンソール8番でgettyを動かす**

```
kill -HUP 1
```

**図12　initプロセスにHUPシグナルを送る**

```
telinit q
```

**図13　initプロセスに/etc/inittabを再読み込みさせる**

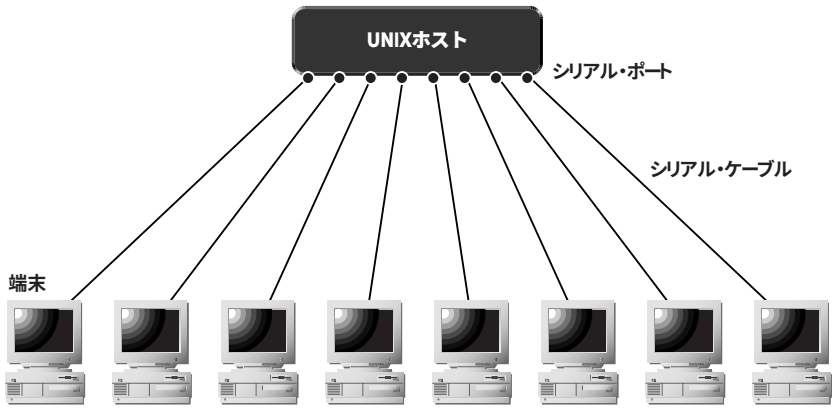


**図14　1台のUNIXマシンを複数の端末で共有**

### シリアル・ポート

シリアル・ポートへの入出力は, デバイス・スペシャル・ファイル/dev/ttyS***n*** (***n*** = 0, 1, 2, ...)を介して行います。例えばシリアル・ポートCOM1は/dev/ttyS0, COM2は/dev/ttyS1といった具合に対応します。シリアル・ポートの場合はコンソールの場合と異なり,ボー・レート, フロー・コントロールやパリティ・ビットなどの通信パラメータを適切に設定しないと通信できませんから, もう少し複雑になります。agettyならば引数に設定値を与えることにより, シリアル・ポートに適切な通信パラメータを設定した上で, ログイン・プロンプトを送出できます。

ただし, 最近はシリアル・ポートにログイン用の端末が直接接続されているケース(**図14**)はまれでしょう。10数年ほど前までは, 1台のUNIXマシンを共有するために複数の端末をシリアル・ケーブルでUNIXマシンに直結して使うのが普通でした。そのころ良く使われたのが米DEC製のVT100端末(**写真1**)です。端末エミュレータでなく, 実物のVT100端末を見たことがある人は今となっては少数派ですね。VT100端末について詳しく知りたい人は, http:

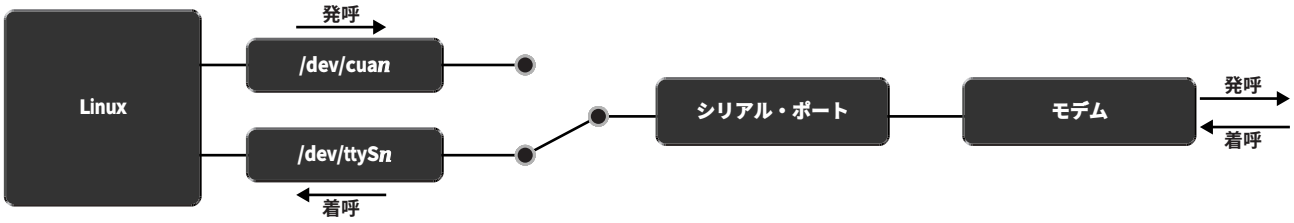


**図15　/dev/cuanと/dev/ttySnのどちらかのみ使用可能**

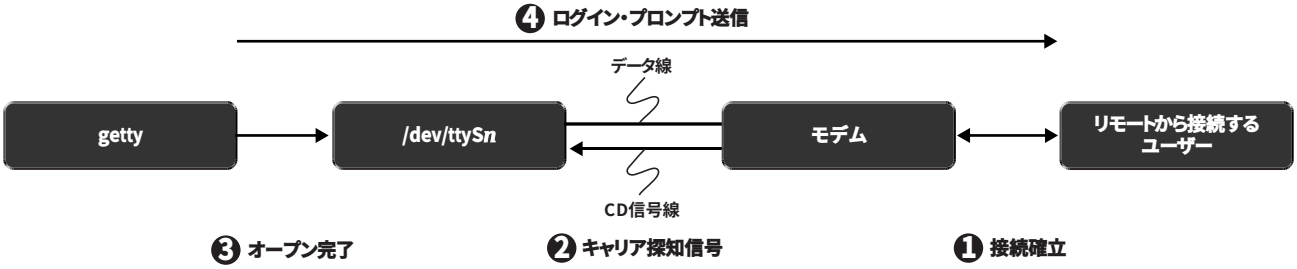


**図16　接続確立からログイン・プロンプト送信までの流れ**

**写真1　VT100端末**

//www.vt100.net/などを参照すると良いでしょう。

ほとんどの場合，シリアル・ポートにはモデムがつながっているのではないかと思います。そうなると通信パラメータだけでなく，モデムの設定も必要になります。大昔のモデム（MODEM）は，名前の通り変調と復調（MOdulation DEModulation）しかできなかったので設定は簡単でしたが，現代のモデムはATコマンドなどで各種設定，発呼/着呼制御が行える上に，FAXモデムであればFAXの発着信までできます。このような多機能モデムが接続されたシリアル・ポート用のgettyがmgettyです。

### /dev/cuanと/dev/ttySn

1つのシリアル・ポートを，着呼（ダイアル・イン）と発呼（ダイアル・アウト）の両方で使おうとすると，少々工夫が必要になります。着呼，すなわちモデムが着信応答した時，速やかにログイン・プロンプトを送信し，電話線経由でユーザーが送信したユーザーIDを受信するためには，常にgettyがシリアル・ポートを監視している必要がありますが，発呼，すなわちローカル・ユーザーがモデムでダイアルして通信するときは，gettyに動いてもらっては困ります。

この問題を解決しようとして，SunOSでは，発呼用のデバイス/dev/cua***n***（***n*** = 0, 1, ...）が/dev/ttyS***n***とは別に作られました。/dev/ttyS***n***と同じシリアル・ポートに対応しますが，Call oUt（発呼）用のデバイスという意味です。もちろん，どちらも同じシリアル・ポートですから，どちらかをオープンしているときは，他方も使用中になり，使えません（**図15**）。

/dev/ttyS***n***はgetty専用になります。すなわち，gettyは/dev/ttyS***n***をオープンしようとしますが，モデムが着信応答してCD（Carrier Detect，キャリア検知）信号線がアクティブになるまでは，ブロックされます（つまり，I/O待ち）。モデムがS0レジスタの設定に従って自動着信応答する（**図16の1**）と，CD信号線がアクティブになり（**同2**），ブロックが解除されて/dev/ttyS***n***のオープンが完了し（**同3**），gettyは直ちにログイン・プロンプトを送信します（**同4**）。

一方，/dev/cua***n***は発呼専用であり，着信応答中でなければ，/dev/cua***n***をオープンしてモデムに接続し，ATコマンドを送ってダイアルできます。/dev/cua***n***は/dev/ttyS***n***と異なり，CD信号線

**＊21　現在普通に売られているモデムはすべて，ATコマンドなどを受け付ける「利口な(smart)」モデムです。**

がアクティブ*22でなくてもオープンできます。モデムが通信先との接続を確立すると，CD信号線がアクティブになるのですが，/dev/cua***n***がオープンされているときは/dev/ttyS***n***のオープンがブロックされ続けるので，gettyが動きだすことはありません。

一方，モデムが着信応答してgettyが/dev/ttyS***n***をオープン中であるときは，/dev/cua***n***がブロックされます。

以上の仕掛けで，着呼と発呼のどちらか一方だけが動き，もう片方はブロックされるという排他制御が実現します。/dev/cua***n***はLinuxでも導入されました。ところが，この方法がうまくいくには，発呼を行うすべてのプログラムが/dev/cua***n***を同じ方法でオープンすることが必要になります。非ブロック・モードでオープンするプログラムがあったり，/dev/ttyS***n***をオープンするプログラムがあったりすると破たんします。

結局のところこの方法は，/dev/cua***n***をロック・ファイル代りに使っているにすぎないのです。ロック・ファイルならば普通のファイルを使う方が良いに決まっています。つまり，シリアル・ポートを使いたいプログラムは，まずロック・ファイルが無いか確認し，無ければロック・ファイルを作ってその他のプログラムがシリアル・ポートを使うのを防いだ上で，/dev/ttyS***n***をオープンすれば良いのです。/dev/cua***n***を使う必然性はありません。

現在のLinuxにも，/dev/cua***n***はありますが，これは単に互換性のために残されているだけであって，使うべきではありません。

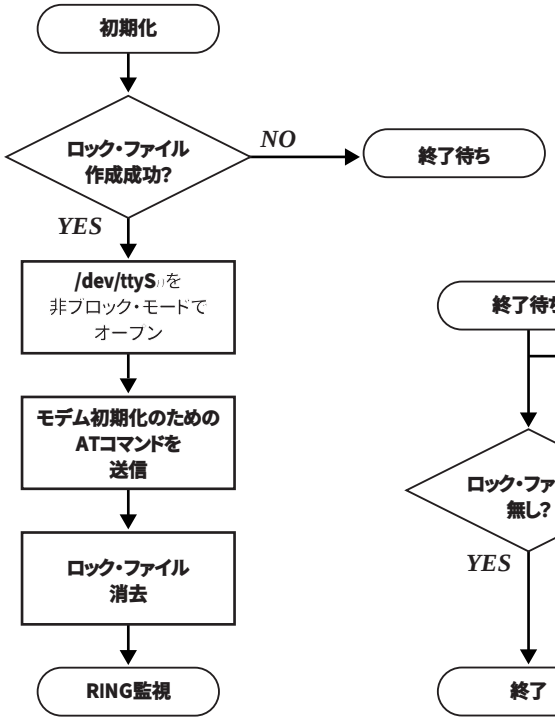


**図17　初期化フロー**

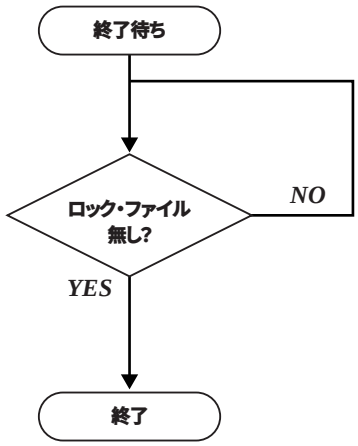


**図18　終了待ちフロー**

## mgetty

mgettyは/dev/ttyS***n***を，非ブロック・モードで常時オープンします。この場合，/dev/cua***n***は使用中ということになり，オープンできません。発呼しようとするプログラムは，まずロック・ファイルが無いかを確認します。ロック・ファイルがあれば，発呼しません。

ロック・ファイルが無ければロック・ファイルを作って，その他のプログラムが発呼しようとするのを防ぎます*23。次に/dev/ttyS***n***を非ブロック・モードでオープンします。つまり，mgettyと発呼するプログラムの両方が同じシリアル・ポートをオープンしていることになります。どちらのプログラムも，シリアル・ポートを等しく読み書きできます。

ただしmgettyは，ロック・ファイルが作られたことを確認すると，ロック・ファイルが無くなるまで，ただひたすら待ち続け，そして終了します。この間，/dev/ttyS***n***に対して一切読み書きしないので，発呼するプログラムの邪魔になることはありません。発呼したプログラムは，/dev/ttyS***n***をクローズするときロック・ファイルを消します*24。するとmgettyが終了して，initプロセスがmgettyの再立ち上げを行います。

mgettyはinitによって立ち上げられると，まずロック・ファイルを作ってから/dev/ttyS***n***を非ブロック・モードでオープンし，モデムに対して各種設定を行うためのATコマンドを送ります（送る内容は，後述するmgetty.config設定ファイルの「init-chat」行で設定できます）。設定が正常に行えたら，ロック・ファイルを消去します（**図17**）。

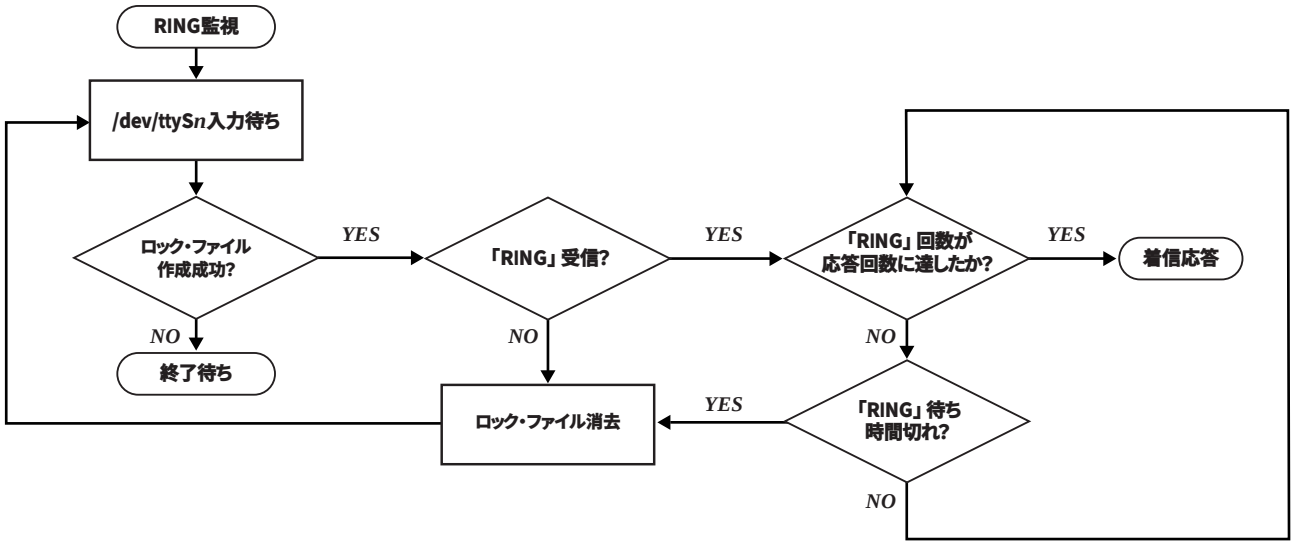


**図19　RING監視フロー**

ロック・ファイルを消すことによって，他のプログラムが発呼できる状態にしつつ，オープンしている/dev/ttyS***n***を監視します。もし/dev/ttyS***n***から受信があれば，まずロック・ファイルを作成できるかを確認します。作成失敗は，その他のプログラムが発呼しようとしていることを意味しますから，前述したようにmgettyは何もせずロック・ファイルが無くなるのを待ち続け終了します(**図18**)。

ロック・ファイル作成に成功した場合は，モデムから届いた文字列を調べ，もし「RING」なら，外部から電話がかかっていることを意味しますからmgettyの出番です。「RING」の回数を数え，あらかじめ設定された回数に達したら着信応答することになります。「RING」が必要回数に達しなかった場合は，途中で電話が鳴りやんでしまったわけですから，ロック・ファイルを消去して，他のプログラムが発呼できる状態にした上で，再び/dev/ttyS***n***の監視に戻ります(**図19**)。

「RING」が必要回数に達した場合，まずモデムに対して「ATA」コマンドを送ります。これは着信応答のためのATコマンドです。つまりmgettyを使用するときは，モデムを自動着信しないモードに設定(S0レジスタに0を設定)する必要があります。

ATAコマンドに対し，モデムから「+FCON」という文字列が返ってきたら，FAX着信があったことを意味します。mgettyはFAX受信モードに移行し，送られてくるFAXデータをスプール・ディレクトリ(通常，/var/spool/fax/incoming)に保存し，終了します。

ATAコマンドに対し，モデムから「CONNECT」という文字列が返ってきたら，データ通信のための接続が確立したことを意味します。ログイン・プロンプトを送信して，ユーザーIDの入力を待ちます。

普通のgettyだと，ユーザーIDを受信したら/bin/loginを実行するのですが，mgettyの場合，受信したユーザーIDによって実行するプログラムを変更することができます。例えば，UUCP専用のユーザーIDならばuucicoを実行し，PPPのフレームが送られてきたらpppdを実行し，それ以外なら/bin/loginを実行する，といった具合です(**図20**)。

ユーザーIDごとの実行するプログラムの設定は，login.configファイルで行います。一例を**図21**に示します。1行目がUUCP専用のユーザー(IDの先頭がU)のための設定，2行目がPPPで接続してきたユーザーのための設定，3行目がそ

**＊22　モデムが通信先との接続を確立するまでは，CD信号線はアクティブではありません。**
**＊23　複数のプログラムが同時にロック・ファイルを作ってしまったらどうするのでしょう?。ロック・ファイルを作るプログラムは，それぞれ自分のプロセスIDをロック・ファイルに書き込みます。だから書いた後でロック・ファイルの内容を確認し，もし自分のIDと異なるIDがロック・ファイルに書かれていたら，競り負けたことになります。自分のIDをロック・ファイルに書き込むことに成功したプロセスのみが発呼することを許されます。**
**＊24　消す前に異常終了してしまっても大丈夫です。ロック・ファイルには，それを作ったプロセスのIDが書き込まれているため，もしそのIDを持つプロセスが存在しないならロック・ファイルは無効となり，mgettyが消去します。**

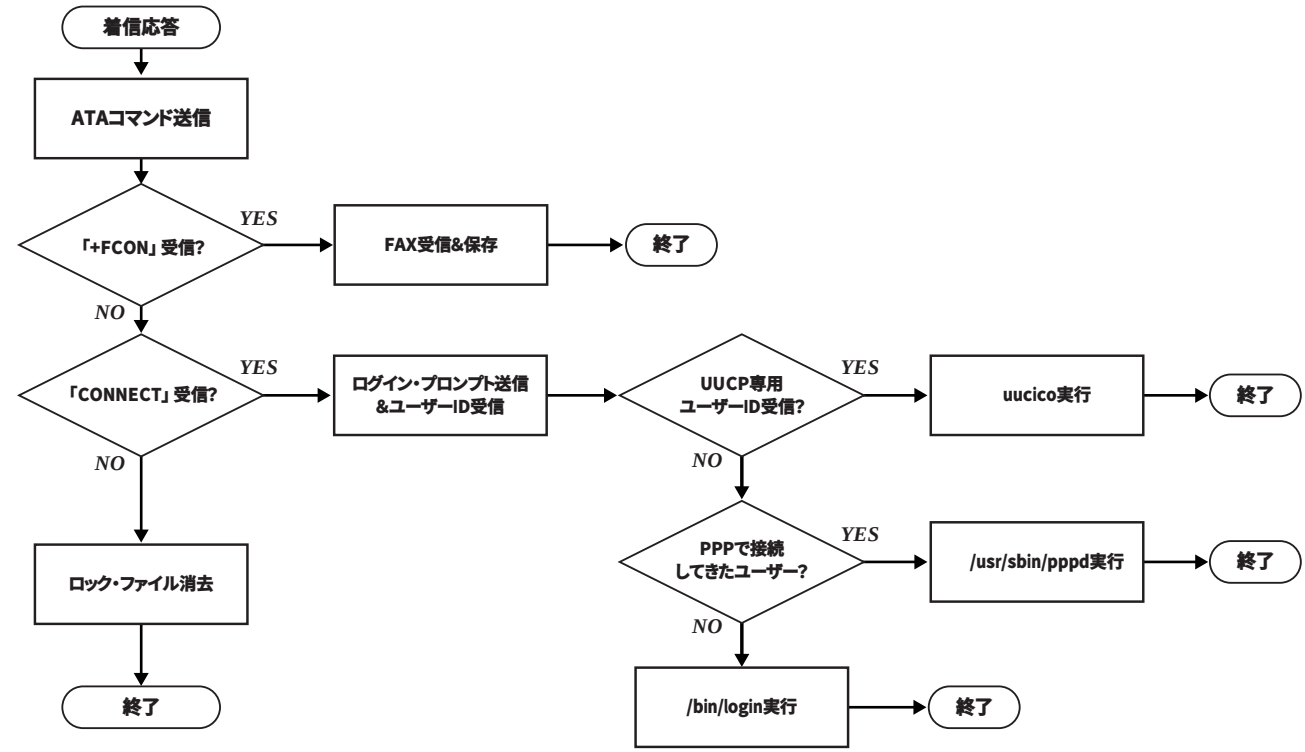


**図20　着信応答フロー**

```
U*              uucp    @       /usr/libexec/uucp/uucico -l -u @
/AutoPPP/       uucp    ppp     /usr/sbin/pppd auth -chap +pap login
*               -       -       /bin/login @
```

**図21　login.config設定ファイル**

```
speed       115200
#switchbd   19200

port-owner  uucp
port-group  uucp
port-mode   0664

fax-id      HIROAKI SENGOKU
fax-owner   fax
fax-group   fax
fax-mode    0660

# MN128SOHO SL11
port ttyS0
  init-chat   "" \d\d\d+++\d\d\dATQ0V1H0 OK ATS0=0E0&D2&C1 OK
  modem-type  data

# MC288XLII
port ttyS1
  init-chat   "" \d\d\d+++\d\d\dATQ0V1H0 OK ATS0=0E0&D2&C1 OK
  modem-type  cls2
```

**図22　mgetty.config設定ファイル**

の他のユーザーのために/bin/loginを実行する設定です。

mgettyの設定の多くは，mgetty.configファイルで行います。一例を**図22**に示します。最初の「port ttyS***n***」行までが，すべてのシリアル・ポートに共通の設定，「port ttyS***n***」行から次の「port ttyS***n***」行までが，それぞれのシリアル・ポートごとの設定です。図22の設定では，/dev/ttyS0はデータ通信専用のモデム（実際はTAですが），/dev/ttyS1はFAXモデムです。

## FAX受信

前述したように，mgettyはFAXを受信するとディレクトリに保存します。しかし，G3 FAXデータそのままの形式で，1ページごとに1ファイルづつ保存するの

で，あまり扱いやすいとは言えないでしょう。そこで複数ページが送られてきた場合は，1つのファイルにまとめ，TIFF形式*25で保存することにします。TIFF形式ならば，Windowsの標準的なアプリケーションである「イメージング」で見ることができます。

mgettyをコンパイルするとき，policy.hにおいて**図23**などのようにプログラムのファイル名を，FAX_NOTIFY_PROGRAMに設定すれば，そのプログラムがFAX受信時に呼び出されます。第1引数に結果コード，第2引数に送信者のID，第3引数にページ数，第4引数以降が受信したFAXデータのファイルがページ数分続きます。

したがって，図23で指定したnew_faxプログラムを，例えば**図24**のようなshスクリプトで実装すれば，FAX着信時にTIFF形式に変換し，/var/spool/fax/tiffディレクトリに「通し番号_送信者ID」という形式のファイル名で保存することができます。また，FAXを受信した旨を，スクリプト1行目の「RECIPIENTS」で設定したメール・アドレスへ知らせます。なお，このスクリプトでは，TIFF形式への変換に，fax2tiffプログラム*26を利用しています。

さらに，/var/spool/fax/tiffディレクトリをファイル/プリンタ・サーバーのSambaを使ってWindowsマシンから共有できるようにしておくと便利でしょう。smb.confの例を**図25**に示します。

```
#define FAX_NOTIFY_PROGRAM "/usr/local/lib/mgetty+sendfax/new_fax"
```

**図23　FAX受信時に呼び出すプログラムの設定（policy.hから抜粋）**

```
#!/bin/sh
RECIPIENTS="sengoku"
SPOOL=/var/spool/fax/tiff
COUNT=/var/spool/fax/tiff.count
PATH=/usr/ucb:/usr/bin:/bin:/usr/local/bin:/usr/local/bin/pbmplus
HangupCode=$1
shift
SenderID=$1
shift
NPages=$1
shift
if [ $# -eq 0 ]
then    exit 1
fi
Count=`cat $COUNT`
base=`echo $1 | sed -e 's/.*\/[0-9A-Za-z]*[-_]*\([^/]*\)\.[0-9]*$/\1/' -e 's/[-_]*$//'`
tiffile="$SPOOL/${Count}_$base.tif"
option=
case `echo $1 | sed 's/.*\///'` in
fn*)     mode="normal"
        option="$option -s"     ;;
ff*)     mode="fine"             ;;
*)       mode="unknown"
esac
fax2tiff $option -1 -o $tiffile -M "$@"
echo 'FAX を受信しました。

        送信者 ID:  "'$SenderID'"
        ページ数:   '$NPages'
        ファイル名: "'$tiffile'"

        モード:     '$mode'
        結果コード: '$HangupCode'

です。
' | mail -s 'FAX received' $RECIPIENTS
expr $Count + 1 > $COUNT
exit 0
```

**図24　FAX着信時にFAXデータをTIFF形式に変換するnew_faxスクリプト**

## FAX送信

mgettyには，FAXを送信するためのsendfaxプログラムが同こんされています。sendfaxの設定ファイルsendfax.configの一例を**図26**に示します。FAXを送信するときのシリアル・ポートなどを設定します。

sendfaxプログラム自体は，G3 FAXデータ形式しか送信できませんが，さまざまな形式のデータをG3 FAXデータ形式に変換して送信用スプールに蓄えるfaxspoolコマンド（sendfaxに付属）を使えば，手軽にFAXを送ることができます。

faxspoolコマンドによって送信用スプールに蓄えられたFAXデータをsend

*25　TIFF Class Fフォーマット。FAXデータを格納するための形式です。
*26　http://www.libtiff.org/を参照。

faxプログラムへ渡す役割を果たすのが，faxrunqdプログラムです。/etc/rc.d/rc.localでfaxrunqdを起動するよう設定しておくと良いでしょう。

### 音声電話と共用する

最近はISDNが普及し，「iナンバー」という，ダイアルイン・サービス(月額900円)に比べれば安価（月額300円）なサービスもあるので，複数の電話番号を手軽に持つことが可能になりました。個人がFAX/データ通信専用の電話番号を持つことも，そんなに特殊なことではなくなったと言えるでしょう。専用の番号があれば，電話を鳴らすことなく自動着信応答が可能ですから，毎日のように自動着信応答させる必要があるなら，専用の番号を確保すべきです。

とは言っても，現在市販されているFAX機のほとんどすべてが，1つの電話番号をFAXと音声通話で併用可能であることからみても，大多数の個人にとっては電話番号は1つだけなのが現状でしょう。そこで，1つの電話番号でFAX/データ通信自動着信応答と音声通話を両立させる方法を考えます。当然，留守番電話が使用できることが必要でしょう。すると自動応答してよいのは留守番電話の方であり，mgettyが応答できる余地が無くなってしまうように思えます。

#### ●Ring Back方式

この問題を解決するのが，Ring Backと呼ばれる方法です。mgettyに応答させるには，まず留守番電話が応答しない範囲で電話を鳴らします。いったん電話を切って，30秒以内に再度電話を鳴らすと，mgettyが応答します。2度目の呼び出しでは，留守番電話より先にmgettyが応答することに注意してください。

Ring Back 方式を利用するには，前述のmgetty.config設定ファイルに図27に示す行を挿入します。

#### ●CNGトーン方式

Ring Backの最大の欠点は言うまでもなく電話を2度かける必要がある，ということです。1回めの呼び出しは電話料金はかかりませんからお金は無駄にはならないのですが，いかんせん面倒です。FAXを送ってくれる人にいちいちRingBackの方法を教えなければいけません。

知人からFAXを受ける場合は面倒くさい方法を強制することも可能ですが，ショップなどにFAXで見積りを送らせる時にRing Backを説明するのは現実的ではありません。

FAXでは，送信側がCNGトーンと呼ばれる1100Hzの音を出します。そこで，留守番電話（あるいは人間）が応答中に電話線を監視して1100Hzの音を検知したら，mgettyがモデムにATAコマンドを送信して応答させるようにすれば，FAX自動着信が実現できます。

電話線の監視は，電話線の音声をPCのサウンド・ボードのマイク端子に入力し，フーリエ変換によって1100Hzのピークを検出することによって行います。詳しい方法は，http://www.gcd.org/sengoku/docs/fax_senseに説明がありますので，興味のある読者は参照してください。

ただしこのページは6年近く昔[*27]のものなので，サウンド・ボードの制御方法が現在のLinuxとは異なっています。mgettyのパッチもそのままでは当たらないと思いますが，フーリエ変換の部分などは参考になるのではないかと思います。

```
[fax]
   comment = incoming FAX
   path = /var/spool/fax/tiff
   public = no
   writable = no
```

**図25　TIFF形式に変換したFAXデータをWindowsから参照（smb.confから抜粋）**

```
fax-devices         ttyS1
max-tries           3
max-tries-continue  y

port ttyS1
  modem-type cls2
```

**図26　sendfax.config設定ファイル**

```
   ringback    y
```

**図27　Ring Back方式を利用するための設定（mgetty.config）**

**＊27　このページを書いた後，FAX/データ通信専用の電話番号を確保したので，この仕掛けが不要になってしまったためです。**