Seeing it the Hard Way

Well, you have been hacking that C program for a month now. You know, the one that is the Swiss Army® knife of UNIX® tools. There isn't a Nobel prize for software but if there was one this tool of yours would win it. Perhaps if you sent it to the executive committee in Stockholm for evaluation they would be moved to establish the new category. Something tells you that they don't accept floppy, tape or American Express®, so you are going to have to print it out so they can read it. Well, why don't you just `pr` all those files to `/dev/lp`? If you are the only one on your system then this should not present a conflict of interests. But, if you share a system with others then your output could be mixed on the page with the person next door if they decide to print their files at the same time. This is the basic problem that the UNIX Line Printing (LP) sub-system solves — it schedules asynchronous events (i.e. print-jobs) to be processed in a sequential manner for a given resource. This basic resource management problem is compounded by printer characteristics (e.g. baud rate, character sets, print-wheels, fonts, PostScript®), media (e.g. pre-printed forms, high-quality paper, envelopes, labels), connectivity (e.g. the printer is attached to another node), and restricted user access. Providing a sub-system to effectively and efficiently manage these resources is a job at which the SVR4 LP sub-system excels.

Printing technology does not seem to be considered a glamorous technology unless it is Display PostScript or Scalable Fonts. But, there are no less than a zillion details associated with printer resource management. This is because the characteristics of the logical and physical devices that control what we may think of as printers can be as diverse as the printers themselves. I have hinted at some of these issues already and although there isn't enough room in an entire year of this magazine to deal with all of them, we will deal with some of the ones that do fit and are useful to the mainstream.

What is a Printer?

This may sound like a silly question, but it is actually a good one. This is because a printer is not always a ''printer.'' A printer can be a printer, like a dot-matrix or PostScript printer, or it can be something really different like a typesetter, pen-plotter, 35mm slide generator, or a one-way FAX card. In fact it could be just about any batch-oriented device. You could even use the LP sub-system to control backup jobs to one of your 9-track tape machines or archive requests to your WORM farm!

What is a Sub-System?

LP consists of two standing servers (or daemons) — `lpsched` and `lpNet` — and several administrative and user commands. `lpsched` is started when the system is booted and is the primary scheduler for printer devices. `lpNet` is started by `lpsched` and is used by `lpsched` to manage network devices and handle the transfer of print-jobs between remote systems. All the commands communicate with `lpsched` using a finite message set and pipe based interprocess communication. The administrative commands are used to configure and control the LP sub-system. They are:

`lpadmin`	is used to define logical printers, their attributes and their accessibility by users.
`lpsystem`	is used to define the communication attributes of remote systems.
`lpfilter`	is used to define filters and their abilities.
`lpforms`	is used to define logical media.
`lpusers`	is used to define print-job priority values to specific users.
`accept`	is used to start the queuing of print-jobs to a printer.
`reject`	is used to stop the queuing of print-jobs to a printer.
`enable`	is used to logically start a printer (i.e. it is now on-line).
`disable`	is used to logically stop a printer (i.e. it is now off-line).
`lpmove`	is used to move print-jobs assigned to one printer to another printer.
`lpshut`	is used to shut down the LP scheduler (`lpsched`).

The user commands are used to submit, monitor and cancel print-jobs. They are:

`lp`	is used to submit files for printing — i.e. it creates a print-job.
`lpstat`	is used to check the status of the entire LP sub-system including print-jobs.
`cancel`	is used to cancel print-jobs.

What's New?

A new feature in SVR4 is the ability to network printers with complete LP semantics. That is, you don't lose control of the print-job once it leaves the client system. Cancelling a print-job from the client will cancel it on the server; stating print-jobs on the client will tell you what is happening to them on the server; and, changing the print-job specification on the client will change it on the server. The new networking capabilities also include SVR4 to BSD connectivity. This allows print-jobs from SVR4 systems to be sent and received from BSD systems. So, if you are slowly migrating from BSD to SVR4, you need not worry about which system is your print server. They already know how to talk to each other.

Furthermore, a BSD user migrating to an SVR4 system will feel right at home if the Compatibility Package (an SVR4 add-on) is installed. In it are the same functionally equivalent commands found on a BSD system. These include

- `lpr`
- `lpc`
- `lpq`
- `lprm`
- `lptest`

PostScript support is also new. The SVR4 implementation comes with filters and drivers that facilitate printing on PostScript printers. As you probably already know, PostScript is a language. A PostScript printer does not accept raw ASCII input. Everything going to a PostScript printer must be in the form of a PostScript job. That is, it must be a PostScript program. The filters provided convert a particular type of output (e.g. ASCII, `ditroff`) into PostScript. The drivers that are provided download fonts and handle communication details with the physical printer.

What Does it Do?

Once printers are defined, queuing and on-line, the most used commands are the user commands. The `lp` command parses its arguments and composes a message that represents the context of the print-job and sends it to `lpsched`. The files that are to be printed are not sent to the LP scheduler, they will be printed from wherever they live. Some notable options available to the user at this point are:

- Print *N* copies of the print-job.
- Notify the user via `mail` or `write` when the print-job has printed.
- Delay the print-job until the specified media (e.g. special pre-printed forms, high-quality paper) is in use on the printer.
- Delay the print-job until the specified character-set or print-wheel is in use on the printer.
- Arbitrarily hold the print-job until released.
- Specify printer-dependent options that are passed on to the interface-program that drives the printer.
- Submit the print-job at a specified priority-level.

Now that the user has submitted a print-job, they can track its progress in the queue, using `lpstat`. The user can also make changes to the queued print-job using `lp`. If the user decides to abort the print-job, they may cancel it, even at the point when it is printing, using `cancel`.

When a print-job makes it to the head of the queue for the printer to which it was assigned, `lpsched` will `fork()` and `exec()` the *interface-program* that drives the printer. The interface-program knows how to control the printer based on its `terminfo` entry that describes its capabilities and printer-model specific information that can't be represented any other place. The SVR4 LP sub-system comes with two interface-programs. The one of interest here is called `standard` and it handles a diverse majority of printers. Since it is a shell-script, it is also intended to serve as an example of the various issues associated with driving a printer, as well as describing the calling interface between it and `lpsched`. This is so that a site could write their own for a non-standard ‘‘printer’’ like a FAX card.

Interface-programs are also responsible for detecting printer faults (e.g. out-of-paper, physically off-line) and reporting these back to `lpsched`. The scheduler then reports these faults to the administrator via some *alert mechanism* (.e.g. mail, write to system console) which is defined when the printer is defined.

It Slices!

The definition of printers and their capabilities is administered using `lpadmin`. In its simplest and complete form for local printers, `lpadmin` is used to specify the logical printer-name, the logical device to which the physical printer is

attached, the `terminfo` entry that contains the definition of the capabilities of the printer, the *content-types* the printer will accept, and the interface-program that drives the printer. For example, to configure a PostScript printer `/dev/lp`

```
lpadmin -p ps1 -v /dev/term/00 -T PS -I PS -m PS
```

This looks confusing, that is because the printer-type (the `terminfo` entry specified with `-T`), the content-type the printer will accept (specified with `-I`), and the interface-program that drives the printer (specified with `-m`), all happen to have the same name. The `terminfo` entry `PS` and the interface-program `PS` are supplied with the SVR4 implementation.

At this point you might ask "If the content-type of the printer is `PS` and the file I want printed is a simple ASCII file, how will it ever print?" (This is the magic part. Watch the hands closely.) LP has a mechanism called *filter pipeline construction* that is driven by a data file called the *filter description table*. This filter-table is administered using the `lpfilter` command. A filter is simply a process that reads the standard-input and converts the data to some new form and writes it to standard-output. A command as simple as `pr` could be a useful filter. `lpfilter` also allows the user to specify the options to be used with a filter based on shell-like regular-expression (i.e. '*', '?', '[a-z]') evaluations of the arguments passed to the `lp` command, when the print-job was submitted.

Based on the filter-table, `lpsched` can dynamically construct filter pipelines to convert one file content-type to another. The filter-table delivered with the system contains the following filter-descriptions

name	*input-type*	*output-type*
postprint	simple	postscript
dpost	troff	postscript
download	postscript	postdown
postio	postdown	PS

`lpsched` reads the filter-table and matching input-type with output-types, dynamically constructs a filter pipeline that converts the given content-type of the file to the desired content-type that the printer can handle using multiple filters if necessary. This pipeline is then passed to the interface-program that drives the printer at the time the print-job is sent to the printer.

In the case of the PostScript printer the ASCII file would be submitted to `lpsched` using

```
lp -dps1 ascii_file
```

The content-type of a simple ASCII file is termed `simple` and the default content-type that the `lp` commands assumes is `simple`. That is, it contains only printable ASCII characters. If the file is of some other type, this is specified using the `-T` *content-type* option to `lp`. In this example, the pipeline would look like

```
postprint | download | postio
```

If the user had submitted a `troff'd` file using

```
lp -dps1 -Ttroff troffd_file
```

the the pipeline would look like

```
dpost | download | postio
```

The `download` filter checks the PostScript output passing through it for use of special fonts that may be stored on-line, which it then downloads into the PostScript stream. The `postio` filter is a special filter in that it doesn't convert its input but instead it knows how to hold a conversation with the PostScript printer. That is, PostScript printers talk back to you when you ask them and sometimes when you don't. A kind-of verbose flow control. The `postio` ''filter'' knows how to respond to this dialogue.

Although this example was PostScript specific the versatility of this mechanism should be obvious. There really is nothing special about filters. In fact this is a basic UNIX system paradigm. And, although the LP sub-system comes with a pre-built filter-table the administrator is free to add new filters as a site requires. There is even an option to `lpfilter` for restoring the filter-table to its original delivered state if it gets corrupted.

It Dices!

LP also provides for the management of logical forms and access control. Now that we have print-jobs printing on our PostScript printer lets assume that the printer is used to print on cheap paper, expensive 20lbs. bond with the company letterhead, envelops and labels. Forms are defined using the `lpforms` command. The definition of a form can include such attributes as the page size and the required character pitch. These attributes get passed to the interface-program that drives the device so that it can prepare the printer to print the data properly on the form. There is also an alignment-pattern attribute. It defines the pattern to print on the printer while aligning pre-printed forms (e.g. checks, invoices, purchase orders).

In this example we will use the simplest form definition which is a comment. In a file called `form_def` we place the following:

```
Comment:  Special PostScript form.
```

Our forms will be called `cheap`, `bond`, `envelop`, and `label`. Now we define the forms using

```
for f in cheap bond envelop label
do
     lpforms -f $f -F form_def
     lpforms -f $f -A mail
done
```

The first `lpforms` command names the form and says use the form definition in file `form_def`. The second `lpforms` command takes advantage of the alert feature. The alert feature notifies the administrator when a print-job that requires a specific form has been queued. The alert type specified is the `mail` type. When the first print-job that requires one of our special forms to be *mounted* on the printer is queued the administrator will be notified by mail. There are other alert types, including any arbitrary shell command.

By default anyone can use each of our forms and our printer. We could restrict access to the printer using

```
lpadmin -p ps1 -u allow:jeffp
```

This would restrict the printer to the local user `jeffp`. This can be useful if the printer is set aside for printing sensitive or proprietary information, or the printer is expensive to operate (e.g. a typesetter) and you don't want users experimenting with it. Since the form `bond` is expensive and it has the company letter head on it, we would like to restrict its use to only users who have a need to use it. This is instead of restricting the entire printer to use by a single user or group of users. This is done by

```
lpforms -f bond -u allow:jeffp
```

Now only the local user `jeffp` can queue jobs requiring the form `bond`.

Now that our forms are defined we need to modify our printer definition to allow those forms to be mounted and to mount the default form. The command

```
lpadmin -p ps1 -f allow:cheap,bond,envelop,label
```

specifies that these form can be used with this printer. Currently we have cheap paper in the paper tray of the printer so the command

```
lpadmin -p ps1 -M -f cheap
```

logically mounts our default form. Print-jobs sent to `ps1` that request the `cheap` form

will now be printed on the cheap paper unless they are specified to need a different form. Print-jobs that do not specify a form are not printed until the special `none` form is mounted. So we don't have to require the user to specify the `cheap` form at all. We can leave the cheap paper in the printer and mount the `none` form. To print a file on the high-quality bond the user would specify

```
lp -dps1 -f bond -T postscript letter.ps
```

If the user submitting the request is `jeffp` then the print-job will be accepted. The `-T postscript` options simply states that the file `letter.ps` contains a PostScript program. The filter pipeline will be shortened accordingly when the print-job is finally sent to the printer.

Since the `bond` form is not currently mounted the administrator is notified by mail that the a print-job has queued that requires the form. The print-job will stay in the queue until the form is mounted. This entails taking the printer logically off-line using `disable`, filling the paper tray with the high-quality bond paper, logically mounting the form using `lpadmin`, and then logically bringing the printer back on-line with `enable`.

Its Over There!

As was mentioned previously, LP is capable of networking with other systems. That is, any system on a network that has a printer attached to it can act as a server for that printer to other systems on the network. Also, any system on the network can be a client of any server. The networking feature requires that you have a network and that it is configured properly. Since this is an article about Printing Technology, network configuration will not be discussed here. But, once your network is installed, networking a client and server LP sub-system is a simple matter.

Since our PostScript printer is already defined, our system will act as the server. Each system needs know about the other. On the server system the command

```
lpsystem -t s5 -T never -R no client
```

adds the system `client` to the list of systems allowed to send print-jobs to the server. It also specifies that the client is a SVR4 (`-t s5`) system, never timeout a connection (`-T never`) and that lost connections should not be retried (`-R no`) — i.e. let the client call the server. On the client system the command

```
lpsystem -t s5 -T never -R 10 server
```

adds the system server to the list of systems to which it can send print-jobs. The

difference here is that the client retries the server after 10 minutes (`-R 10`) if the connection is lost. The server printer is already configured. The client defines the same forms locally and incants

```
lpadmin -p remote_ps -s server!ps1 -T PS \
     -I simple,troff,postscript \
     -f cheap,bold,envelop,label
#
#  start queuing print-jobs to local printer 'remote_ps'
#
accept remote_ps
#
#  start sending print-jobs to remote printer 'ps1' on system
#  'server'
#
enable remote_ps
```

and all the capabilities of the printer `ps1` that a user on the server has are now available to a user on the client. The content-type specification

```
-I simple,troff,postscript
```

forces filtering to occur on the server system.

You Can Even Get To It From Your Application

At this point it should be clear why you don't want to `cat`, `pr`, or `write()` directly to `/dev/lp`. But, I'll state is again.

1. `/dev/lp` may be busy but not locked.
2. `/dev/lp` is not always `/dev/lp`. Sometimes it is `/dev/term/00`, a serial device with all the hassles of baud-rate and special character processing. `/dev/lp` is not always connected to a friendly, simple ‘‘printer.’’

If you want to get at printers through an application — i.e. via C code — use the Section 3S `popen()` or `system()` functions. The `popen()` function creates a bi-directional pipe between your process and the shell command-line you give it. Using it you can write the data you want printed directly into the `lp` command. For example:

```
int
PrintData (char *destp, char *bufp)
{
        char  cmdbuf [128];
        FILE *pipep;

        (void) sprintf (cmdbuf, "/usr/bin/lp -d%s", destp);

        pipep = popen (cmdbuf, "w");

        if (! pipep)
                  return    0;

        (void)    fprintf (pipep, bufp);

        (void)  pclose (pipep);

        return    1;
}
```

Using the `system()` requires that you already have the data you want printed in a file on disk. For example,

```
int
PrintFile (char *destp, char *pathp)
{
        char  cmdbuf [128];

        (void) sprintf (cmdbuf, "/usr/bin/lp -d%s %s",
                  destp, pathp);

        return    system (cmdbuf);
}
```

It is a Swiss Army Knife!

The LP sub-system in SVR4 is rich with functionality. In fact, we have just skimmed the surface of its most useful features. So, before you re-invent the wheel for your application, look and see what the system is already providing you.

Acknowledgements

I would like to thank Carey Hines for his help in reviewing this article.

Bibliography

[1] *UNIX System V Release 4: System Administrator's Guide,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[2] *UNIX System V Release 4: System Administrator's Reference Manual,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[3] *UNIX System V Release 4: User's Guide,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[4] *UNIX System V Release 4: User's Reference Manual,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[5] *UNIX System V Release 4: Programmer's Reference Manual,* Prentice Hall, Englewood Cliffs, NJ, 1990.