# PAS: A Printer Accounting System for Linux

Murali Thiyagarajah
School of Computer Science, Carleton University
Ottawa, Ontario K1S 5B6, CANADA

22 June 1995

**Abstract**

There are many hardware solutions for accounting printer usage in a networked environment such as Linux. This document describes a software solution for printer accounting on a PostScript printer using an RS-232 serial interface on a Linux server.

## 1 Introduction

The Printer Accounting system or PAS was implemented at the School of Computer Science, Carleton University in order to regulate the printer usage by students. PAS uses an RS-232 serial interface and has been tested on a Linux server running Linux kernel version 1.1.64. Porting to other platforms has not been attempted but is possible with very few changes to other 4.3 BSD based systems.

### 1.1 Overview of PAS

The main engine of PAS is a program called `pas`, which implements the communication with the PostScript printer through a bidirectional serial line and performs the accounting. It uses an input filter called `pas-if`, to differentiate a PostScript file from a plain text file.

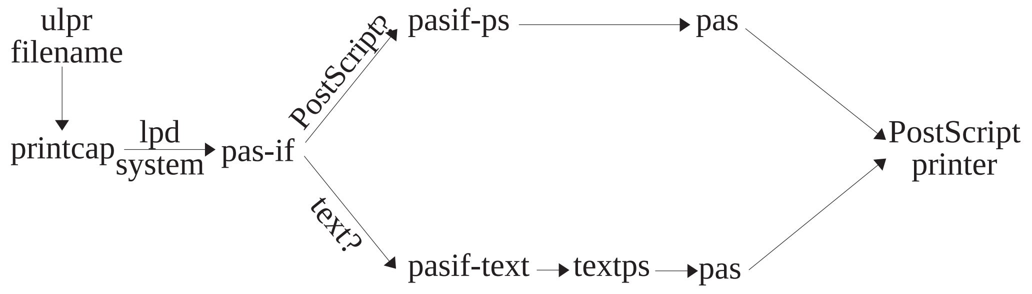

Figure 1: Overview of PAS

The main feature of PAS is implementing the accounting with a quota system. Every printer user is assigned a page quota. After each print job the database will be updated to reflect the number of pages printed. The user is also sent a mail message with the number of pages printed and the remaining quota in his/her account. Once the quota is reached PAS will not allow new print jobs from the user's account. A separate file for each printer user is maintained in a directory named `quotas` under the spooling directory. This file contains four fields, namely, current page count, page quota, printer password, and mailing address. If no mailing address is provided daemon will attempt to forward all the messages to the user's account from where the print job originated. Current page count is initially zero and incremented with each print job. Page quota and printer password are assigned at the time of printer account creation by either the `mkpasusr` utility. (Refer to section 3.4).

In order to prevent anyone trying to communicate directly with printer through the `lpd` daemon to send print jobs, a password security mechanism has been built in. A 60 character password string, maintained in the file `.pp` under the user's home directory, is compared with the user specific password string maintained by the print daemon in the spooling directory before print job is spooled to the printer. Password string from the user's `.pp` file is transmitted to the daemon for verification along with the print job. A modified `lpr`, called `ulpr`, implements this task by prepending the password string to the print file before sending it to the daemon. In addition to the password string, there are several other information such as version number, flags, user name, job title, host machine, are transmitted to the daemon by `ulpr`. The `pas-if` filter stripes off all the above data from the actual print job before sending the print job to `pas` for processing.

The PAS obtains the number of pages printed from each job by taking the difference between the final page count and the start page count from the PostScript printer's ROM. A power failure or a printer shutdown before a print job is completed may potentially cause errors in the accounting and lead to possible abuses. In order to prevent this, PAS logs the user's id at the start of the print job. It also maintains a record of page count from the last successful print job. At the start of each print job, the current page count from the printer's ROM is compared with the page count from the last successful print job. If there is any discrepancy, then the last printer user's account will be debited with the difference.[1] The PAS also deals intelligently with the output generated by the printer: printer errors (such as "out of paper") are logged using syslog; other errors and output generated by the user's job are mailed back to the user. Status file (displayed by ulpq) is always updated to reflect printer's current status.

PAS supports banner printing. If you wish to have banner printing, then you should add `of=/usr/local/lib/pas-of`, and remove the `sh` capability from the `/etc/printcap` file. You will probably also want to modify the banner printing code in `banner.ps`. Note that `pas-of` requires the `sb` capability in the printcap file. If your printer stacks face up and you are using a 4.3 BSD (or later) compatible lpd, then you can add an `hl` capability to the printcap file to specify that the banner page should be printed at the end of a job instead of at the beginning.

## 1.2 Features of PAS at a Glance

(1) Secure printer accounting through RS-232 interface:

   (a) When a user's account file is updated the user is notified through an e-mail. If desired, user can request the messages to be sent to a different e-mail address.

[1] A page count difference of $\leq 5$ is ignored to account for a few start-up pages.

    (b) Accurate database updates even in the case of power failures or printer shutdowns.
    (c) PAS cancels printing when print quota exceeds even in the middle of a print job.
    (d) Use of printer password for added security.

(2) PAS acts as an input filter preventing anyone from communicating with the printer through `lpd` daemon.

(3) Maintains a trace file for use by system administrators for the purpose of resolving account disputes.

(4) Supports banner printing.

(5) Automatically performs page reversal of the output for the benefit of printers that stack face-up. Page reversal filter supports version 3.0 of Document Structuring Conventions (DSC) of PostScript.

(6) Uses a text to PostScript program (`textps`) which supports the ISO Latin-1 convention.

(7) Design of PAS allows a seamless integration for printing on to a Linux network printer from other platforms such as Windows and Mac.

## 2 Description of PAS

The first thing one needs to realize about printing on a PostScript[2] printer is that we don't send a file to the printer to be printed - we send a PostScript program which describes the page layout of one or more pages to the printer for execution. There is usually a PostScript interpreter within the printer that executes the program generating the printed output. So when someone says that he is printing a file on a PostScript printer that means he is executing a PostScript program which in turn will produce a printed output on the print engine. We can print (i.e: execute) a PostScript file by copying

[2]The PostScript language is a programming language designed to convey a description of virtually any desired page to a printer. For more info on PostScript language programming see [1], [2], and [3]

| Char | Octal value | Description of Function |
|---|---|---|
| Control-C | 003 | Interrupt. Causes the PostScript `interrupt` operator to be executed. Normally this terminates the PostScript program being executed. |
| Control-D | 004 | End of file. |
| Line feed | 012 | End of line, the PostScript new line character. If a return and line feed are received in sequence, only a single newline character is passed to the printer. |
| Return | 015 | End of line. Translated to PostScript newline char. |
| Control-Q | 021 | Start output (XON flow control). |
| Control-S | 023 | Stop output (XOFF flow control). |
| Control-T | 024 | Status query. The PostScript interpreter responds with a one-line status message. |

Table 1: Special characters sent from host to PostScript printer

it directly (`cp filename /dev/ttyS0` or `cat filename > /dev/ttyS0`) on the serial port to which the PostScript printer is connected. If it is a plain text file then we need to translate it into a PostScript program before sending it to the printer. So in the simplest situation one can directly copy a file to the serial port and pick up the output at the printer. But what do we do if we want to have control over who can access the printer and how much can he or she print on it? To do that we need to be able to intelligently filter the input file and communicate with the PostScript printer. PAS provides an intelligent way of filtering the input files and communicating with the printer.

Before we describe the functions that form PAS, we need to tell some more information about communicating with a PostScript printer. A PostScript printer (or an interpreter) sends and receives seven-bit ASCII characters. A PostScript program consists entirely of printable ASCII characters. Some of the nonprinting ASCII characters have special meaning, as shown in Table 1.

The PostScript end-of-file character (Control-D) is used to synchronize the printer with the host. We send a PostScript program to the printer and then send an EOF to the printer. When the printer has finished executing the PostScript program, it sends an EOF back. While the interpreter is executing a PostScript program we can send it an interrupt (Control-C). This normally causes the program being executed by the printer to terminate.

The status query message (Control-T) causes a one-line status message to be returned by the printer. All messages received from the printer have the following format:

```
%%[ key: val ]%%
```

Any number of *key: val* pairs can appear in a message, separated by semicolons. The status messages have the form

```
%%[ status: idle ]%%
```

Other status indications, besides `idle` (no job in progress), are `busy` (executing a PostScript program), `waiting` (waiting for more of the PostScript program to execute), `printing` (paper in motion), `initializing`, and `printing test page`.

A printer error is indicated by a message of the form

`%%[ PrinterError:` *reason* `]%%`

where *reason* is often `Out of Paper, Cover Open,` or `Miscellaneous Error`. After an error has occurred, the PostScript interpreter often sends a second message

`%%[ Flushing:` *rest of job (to end-of-file) will be ignored* `]%%`

To handle these messages we have to parse the message string, looking only at the characters within a pair of the special sequences `%%[` and `]%%`. A PostScript program can also generate output from the PostScript `print` operator. The special characters that are sent by the PostScript interpreter to the host computer are listed in Table 2.

| **Char** | **Octal value** | **Description of Function** |
|---|---|---|
| Control-D | 004 | End of file. |
| Line feed | 012 | Newline. When a newline is written to the interpreter's standard output, it is translated to a return followed by a line feed. |
| Control-Q | 021 | Start output (XON flow control). |
| Control-S | 023 | Stop output (XOFF flow control). |

Table 2: Special characters sent from PostScript printer to computer

## 2.1 The Main Program

In this section we explain the different functions that make up PAS. We first start with the `main` function as the execution begins there. A pseudocode of the main program is given below.

**Algorithm 2.1**

```
begin
  process_commandline_args;
  get_status;
  if (passwd_ok)
    then get_page(start_page);
         handle_error;
         check_quota;
         if (current_pagecount < quota)
           then send_file;
                get_page(end_page);
                update_quota;
           else
                if (current_pagecount = $-\infty$)
                  then mail_usr('Cannot print: no page quota set')
                  else
                     mail_usr('Cannot print: quota exceeded') fi
           else
                mail_usr('Cannot print: unknown user'); fi; fi;
end
```

Command-line arguments are first processed, many of which (see the section of `main()` which does this) can be ignored for a PostScript printer. We then fetch the printer status to assure that it is ready by calling `get_status`. If the printer is ready then we verify the user's printer password by calling `password_ok`. If the user has a valid PAS account then we get the printer's starting page count by calling `get_page`. We then call the function `handle_error` to verify that PAS updated the last user's account file correctly (See description of `handle_error` for more information). We then call `check_quota` to check the current page quota of the user to decide whether the user can be allowed to print the current job. If the current page count (`quota_pagecount`) is less than the allotted quota, then we send the file (i.e: the PostScript program) to the printer using `send_file`, get the printer's final pagecount using `get_page`, update the accounting file in the `quotas` directory by calling `update_quota` and terminate.

## 2.2 The `get_status` Function

After processing the command-line arguments, the main program calls the `get_status` function. The `get_status` function fetches the status by sending a Control-T to the printer (see Table 1). The printer should respond with a one-line message. If the printer is ready for a new job, then the message from the printer will have the following form:

```
%%[ status: idle ]%%
```

This message is read and processed by the function `proc_some_input`, which will be described later.

If the printer is waiting for us to send it more data for a job that it is currently printing, then the message will be as follows:

```
%%[ status: waiting ]%%
```

This means something funny happened to the previous job. To clear this state we send the printer an EOF and terminate. A pseudocode of the `get_status` function is given below.

**Algorithm 2.2**

```
This function obtains the status from printer
begin
  set_alarm(5);
  tty_flush;
  send_Ctl_T;
  while (status = INVALID) do
        proc_some_input; {wait for something from printer }
        if status = IDLE
           then clear_alarm;
                return; fi;
     else if status = WAITING
           then send_EOF;
                sleep(5);
                exit(EXIT_REPRINT); fi;
     else if status = BUSY
           then return; fi;
     else if status = UNKNOWN
           then sleep(10);
                exit(EXIT_REPRINT); fi; od;
end
```

## 2.3 The `passwd_ok` Function

If the printer is ready for a new job then the main program calls the `passwd_ok` function to verify the password of the print job. The print command, `ulpr`, prepends the user's printer password (along with some other information) to each print job. This printer password is compared with a user's password kept in the `quotas` directory before the job is allowed to print.

## 2.4 The `get_page` Function

Once the password of the user is verified the main function calls the `get_page` function to obtain the current page count (`start_page`) from the printer's ROM. The `get_page` function uses a small PostScript program to read the current page count. Although we could fetch the page count and print it using just

```
statusdict begin pagecount end = flush
```

we purposely format the output to look like a status message returned by the printer:

```
%%[ pagecount:  N ]%%
```

By doing this the function `proc_some_input` handles the message similar to any other printer status message.

**Algorithm 2.3**

```
1  This function obtains the current pagecount from printer
2  begin
3     init_pagecount-string; { initialize pagecount string }
4     set_alarm(30);
5     tty_flush;
6     send_pagecount_string_to_printer;
7     proc_some_input; {read results from printer }
8     send_EOF;
9     proc_upto_EOF; {wait for EOF from printer}
10    clear_alarm;
11 end
```

## 2.5 The `handle_error` Function

One of the potential problems with a printer accounting system such as ours is power failures in the middle of a print job. To ensure that the user's printer accounting file is updated correctly in the case of a power failure, the `handle_error` function logs the id of the user (in the file `last_usr`) at the beginning of every job. PAS also maintains printer's last page count in the file `last_pg`. This is the printer's page count from the last job[3]. When a new job arrives, `handle_error` compares the start page count (`start_page`) with the number stored in `last_pg`. If `start_page` > `last_pg` then the difference is debited from the last user's account. Since the new job is started only after the previous user's account file is updated, the `handle_error` function

[3]The page count maintained in `last_pg` is always accurate regardless of whether the last job is a successful one or not. How?

works correctly even in the case of multiple successive power failures. The `handle_error` function also sends an e-mail to the last user (copy to printer managers) whenever his or her account file is updated in such a situation.

## 2.6 The `check_quota` Function

The `check_quota` function is called by the main program to obtain the current page count for the user. The current page count is read from the user's accounting file and stored in the variable `quota_pagecount`. The `quota_pagecount` is always less than the user's `quota`.

## 2.7 The `send_file` Function

The function `send_file` is called by `main` program to send the user's PostScript program to the printer. This function is just a `while` loop that reads from the standard input (`getchar`) and calls the function `out_char` to output each character to the printer. When the end of the file is encountered on the standard input, an EOF is sent to the printer (indicating the end of job), and we wait for an end of file back from the printer (`proc_upto_eof`).

**Algorithm 2.4**

```
This function sends the user's PostScript program to printer
begin
  set_intr(); { we catch SIGINT }
  while ((c = getchar())! = EOF) do
        out_char(c); { output each character }
  od
  out_char(EOF); { output final buffer }
  send_EOF; {send EOF to printer }
  proc_upto_EOF; {wait for printer to send EOF back }
end
```

## 2.8 The `out_char` Function

The function `out_char` is called by `send_file` to output each character to the printer. When the argument to `out_char` is `EOF`, that's a signal that the end of the input has been reached, and the final output buffer should be sent to the printer.

The function `out_char` places each character in the output buffer, calling `out_buf` when the buffer is full. We have to be careful writing `out_buf`: in addition to sending output to the printer, the printer can be sending us data also. To avoid blocking on a `write`, we must set the descriptor nonblocking. We use the `select` function to multiplex the two I/O directions: input and output. We set the same descriptor in the read set and write set. There is also a chance that the `select` can be interrupted by a caught signal (`SIGINT`), so we have to check for this on any error return.

If we receive asynchronous input from the printer, we call `proc_input_char` to process each character. This input could be either a status message from the printer or output to be mailed to the user. When we `write` to the printer we have to handle the case of the `write` returning a count less than the requested amount.

## 2.9 The Functions that Process Printer Messages

The functions that process the message that was accumulated by the input functions given above are defined in the file `message.c` (Please see the program in `pas1.0.tar.Z` for details of the function). The function `msg_init` is called after the sequence `%%[` has been seen, and it just initializes the buffer counter. `msg_char` is then called for every character of the message.

The function `proc_msg` breaks up the message into separate *key: val* pairs, and looks at each *key*. The ANSI C function `strtok` is called to break the message into tokens, each *key:val* token separated by a semicolon. A message of the form

```
%%[ Flushing: rest of job (to end-of-file) will be ignored ]%%
```

causes the function `printer_flushing` to be called. It flushes the terminal buffers, sends an EOF to the printer, and waits for an EOF back from the printer.

If a message of the form

```
%%[ PrinterError: reason ]%%
```

is received, `log_msg` is called to log the error. If a status message is returned, denoted with a *key* of `status`, it is probably because the function `get_status` sent the printer a status request (Control-T). We look at the *val* and set the variable `status` accordingly.

### 2.10 The `update_quota` Function

The `update_quota` function is called by the main program after obtaining the final page count from the printer to update the account file of the user.

## 3 Installing and Configuring PAS

### 3.1 Installing PAS

Installing PAS on your Linux network is easy. Just uncompress and untar the file `pas1.0.tar.Z`. First run `make`. Become `root` and then run `make install`. This moves the files `pas`, `pas-if`, `pasif-ps`, and `pasif-text` to `/usr/local/lib` and `psref`, `textps`, `ulpr`, `ulprm`, `ulpq` and `mkpasusr` to `/usr/local/bin`. Once you have installed PAS you need to configure your printer entry on the `/etc/printcap` file.

### 3.2 Configuring Printer Entry on `/etc/printcap`

Before sending a job through PAS first try to cat a file directly to your printer port by using `cat file.ps > /dev/ttyS0` (in my case). Note that your file must be a PostScript file. If it doesn't print, then you probably don't have the correct stty settings on your serial port. Take a look at the stty settings on the port by typing `stty -a < /dev/ttyS0`. You will need to change the settings on some flags before being able to cat a file successfully to the port. Here is what `stty -a` looks like for my printer port:

```
speed 38400 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <undef>;  eol2 = <undef>; start = ^Q; stop = ^S;
```

```
susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr
-igncr icrnl ixon -ixoff -iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel
nl0 cr0 tab0 bs0 vt0 ff0
-isig -icanon -iexten -echo -echoe -echok -echonl -noflsh
-xcase -tostop -echoprt -echoctl -echoke
```

The only changes between this and the way the port is initialized at bootup are `-clocal, -crtscts,` and `ixon`. Your port may well be different depending on how your printer does flow control.

Once you have your stty settings right, you need to translate from `stty` settings to printcap flag settings. There are two sets of flags which you will need to set, plus the baud rate (Note: the `fc` flag setting seems to override the `br` capability, so be sure to set that correctly as well as the `br`. Each of the flags can have bits set and cleared. Clearing is done first, so specify the clear flags (`fc` and `xc`) before the set flags (`fs` and `xs`). Setting the `br` capability is self-explanatory. Example: `br#38400`.

In order to set the `fc, xc, fs` and `xs` flags, you need to look at the file `termios.h` in the directory `/usr/src/linux/include/linux`. Go to the section that starts out

```
/* c_cflag bit meaning */
#define CBAUD     0000017
```

This section lists the meaning of the `fc` and fs bits. You will notice that the names there (after the baud rates) match up with one of the lines of stty output. Note which of those settings are preceded with a - in your stty output. Sum up all those numbers (they are octal). This represents the bits you want to **clear**, so the result is your `fc` capability. Of course, remember that you will be setting bits directly after you clear, so you can just use `fc#0177777`.

Now do the same for those settings (listed in this section) which do not have a - before them in your stty output. In my example the important ones are

CS8 (0000060), HUPCL (0002000), and CREAD (0000200). Also note the flags for your baud rate - mine is 0000017. Add those all up, and in my example you get 0002277. This goes in your `fs` capability (`fs#02277` works fine in my case).

Do the same with set and clear for the next section of the include file, `c_lflag bits`. In my case I didn't have to set anything, so I just use `xc#0157777` and `xs#0`.

Before testing PAS on your system, create a very simple input filter called `if` as a shell script.

```
#!/bin/sh
cat > /dev/ttyS0
```

Move the above file to `/usr/local/lib` and do `chown root.root` and `chmod +x if`. Now add the following lines to your `/etc/printcap` file and try to print a **PostScript** file.

```
lp:\
:br#38400:lp=/dev/ttyS0:\
:sf:sh:rw:\
:fc#0177777:fs#02277:xc#0157777:xs#0:\
:af=/var/adm/psacct:lf=/var/adm/pslog:sd=/var/spool/pslpd:\
:if=/usr/local/lib/if:
```

Your configuration should now work[4]. Now that you are certain that you have the correct flag values in the printcap file, change the `if` entry to the actual input filter, `:if=/usr/local/lib/pas-if:`.

## 3.3 Quota directory and Other Files

You need to create a directory named `quotas` in the spooling directory (in my case `/var/spool/pslpd`). Make sure that it is owned by daemon, group daemon, and are user and group writable (`drwxrwx---`).

[4]For installing and configuring `lpd` see [4]. The latest `lpd` can be found in `tsx-11.mit.edu/pub/linux/packages/net/new-net-2/lpd-590?.tar.gz`.

```
mkdir /var/spool/pslpd/quotas
chown daemon.daemon /var/spool/pslpd/quotas
chmod ug=rwx,o= /var/spool/pslpd/quotas
```

Also you need to create the files, `printer_manager` and `tray_capacity` in the spooling directory and set their permissions to `644`. The first file contains the e-mail addresses of two system admin people who will be notified by PAS in case of any problems, usually `paper out` errors. The second file maintains the total paper capacity of printer trays. This number is used by PAS to determine when printer runs low on paper. My `/var/spool/pslpd/printer_manager` file has two entries as shown below.

```
elliott@louie
murali@louie
```

My `/var/spool/pslpd/tray_capacity` file has just one entry, `750`, indicating the total tray capacity of my printer.

To access PAS you need to create a PAS account. This is done by creating a PAS user list file (`/var/spool/pslpd/userlist`) and running the `mkpasusr` program (Refer to section 3.4 for more information on creating PAS accounts). Create a file called `userlist` in the spooling directory and make just one sample entry by typing your login-name in it. Run `mkpasusr` now and your PAS account will be created.

Now that you have configured the `printcap` file, created the `quotas` directory, other necessary files, and a PAS account for yourself, you can test your setup by printing a file (`ulpr <filename>`).

## 3.4 Setting up PAS accounts

In order to allow users to access a printer on a linux network connected through PAS , you need to create PAS account for each user with the `mkpasusr` utility. The `mkpasusr` utility is a perl program. It assumes that your spooling directory is `/var/spool/pslpd`. So you may need to change the spooling directory in the `mkpasusr` script according to your setup. Create a file named `userlist` in the spooling directory with all `PAS` users' Linux login names, one login name per line with no empty lines at the beginning of the file. The default print quota is 1000. If you need to set another quota value, you need to pass it as a command-line argument (eg. `mkpasusr -q2000`) or edit the `mkpasusr` program.

Run `mkpasusr` as root to create the PAS accounts for all users in the `userlist` file. Make sure that you run `mkpasusr` from a machine with write access to the `var/spool/pslpd` directory and all the users' home directories. If there is any error in creating a user's PAS account, it will be logged in the file `mkpasusr.err` in the spooling directory.

When you use PAS for the first time, you need to update the file `last_pg` with the current page count (Current page count can be obtained by printing a test page from the printer). You also need to edit the command scripts, `ulpr, ulprm,` and `ulpq` to change the printer name. The default printer name is `lp`.

## 3.5 Messages Generated by PAS

PAS logs some messages pertaining to each print job on to the files `lpd-errs` and `syslog` in directory `/var/adm`. These files must be examined in case of any accounting discrepancies (eg. when a user reports his quota has been used up without his knowledge). The messages in these files are self- explanatory. PAS also sends the following messages to the user depending on the status of the last print job.

(1) **Subject: Printer job on printer ug**
Job printed from: louie
Job requested on: Wed Jun 21 14:27:43 1995
No of page(s) printed: 30
Average thru.put: 7.79 pages/min
Remaining print quota: 1740/2000.

(2) **Subject: Printer job on printer ug**
Cannot print: printer account not setup. See system admin folks for help.

(3) **Subject: Printer job on printer ug**
Cannot print: print quota exceeded.

(4) **Subject: Printer job on printer ug**
Print quota reached. Your print job may not be complete!
Job printed from: louie
Job requested on: Wed Jun 21 15:17:13 1995
No of page(s) printed: 11
Average thru.put: 6.21 pages/min
Remaining print quota: 0/2000.

(5) **Subject: Printer job on printer ug**
There was a probable power failure while your last print job from `alpha07` requested on Wed Jun 21 14:27:43 1995 was printing. The total number of pages printed was 32. Your print account file has now been updated.

# 4 Printing from Windows Applications

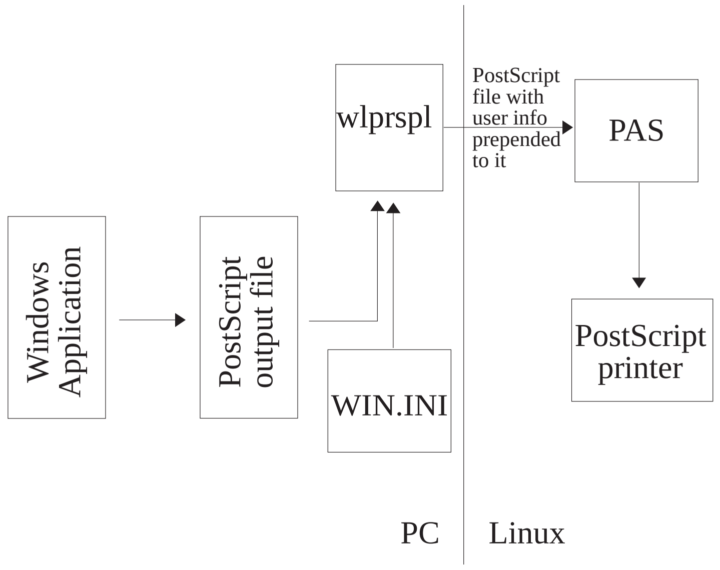

Figure 2: Printing from Windows Applications on to PAS

Printing from Windows applications to a printer connected through PAS on a Linux network is easy, thanks to Thomas Heil's Windows print spooler package, `wlprspl`. You need to send an e-mail to `th.heil@kfa-juelich.de` to obtain the version of `wlprspl`[5] which can work with PAS. Read the `install/readme` files that came with `wlprspl` and install it on your PC. After installing `wlprspl` you need to do the following:

(1) Edit your `WIN.INI` file located in the `WINDOWS` directory and add the following section to it.

```
[PAS]
VersionNo=1.0
Flag=
UserName=
```

[5]the versions of `wlprspl` that are available in standard ftp sites will not work with PAS.

**PP=**
**JobTitle=**
**MachineName=**

Now fill in the **UserName** field with your login name on the Linux network. If you have a PAS account on your Linux network (you need to have one in order to get printing to work from your PC), you will have a file named .pp in your home directory on Linux. The file .pp contains your printer password string. Copy this password string on to the **PP** field of the [**PAS**] section. You need not fill the other fileds of [**PAS**].

(2) Now go to the [**WSOCKAID**] section of the `WIN.INI` file. Fill in the **User** field with your login name on Linux network. Fill in the **LPRQueue** field with the printer name that you would use to access PAS from Linux. Also fill in the **LPRServer** filed with the Linux machine which drives the printer. If you're in doubt check with your system administrator. Your [**WSOCKAID**] section will now look something like below:

```
[WSOCKAID]
User=your_login_name_on_Linux
Host-Name=PC565_003
Domain=scs.carleton.ca
LPRQueue=ug
LPRServer=omega
```

Save the `WIN.INI` file and then exit Windows. Restart Windows to put the changes into effect. Startup `wlprspl` by clicking on to its icon and you should now be able to print to your Linux printer through PAS from any Windows application.

## 5 Conclusion

PAS described in this technical report is a very secure printer accounting system for Linux or other 4.3 BSD based network systems. It implements printer accounting through a page quota system which is strictly enforced by a two-way communication with the PostScript printer. In addition to users' login ids, printer passwords are used for added security. By providing a seamless interface, it makes printing from other platforms such as Windows or Mac an easy task. The security issues are completely hidden from network users (until of course when they receive messages from PAS).

One subtle problem (though will not cause any errors) with the current version of PAS is, if a power failure or a printer shutdown occurs while PAS is printing a job, then the user's account file is updated only when the next job arrives. On its next version, I hope to improve PAS so that it will update the user's account file as soon as a power failure or a printer shutdown occurs. Also PAS ,as implemented now, will only handle either plain ascii text files or PostScript files. The `pasif-ps` input filter needs to be enhanced to handle many other common input file formats such as *.dvi.

## References

[1] Adobe Systems Inc. *PostScript Language Tutorial and Cookbook.* Addison-Wesley, Reading, Massacusetts, 1985.

[2] Adobe Systems Inc. *PostScript Language Reference Manual.* Addison-Wesley, Reading, Massacusetts, 1986.

[3] Adobe Systems Inc. *PostScript Language Program Design.* Addison-Wesley, Reading, Massacusetts, 1988.

[4] Grant Taylor and Brian McCauley. *Linux Printing HOWTO Guide.* Linux Document Project, 1994.