The Implementation of the LP Spooling System

J. R. Kliegman

Bell Laboratories
Piscataway, New Jersey 08854

1. INTRODUCTION

LP is a system of commands that performs diverse spooling functions under the UNIX† operating system. Because its primary application is off-line printing, this paper focuses mainly on spooling to line printers. LP allows administrators to customize the system to spool to a collection of line printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users are provided the capabilities of queuing and canceling print requests, preventing and allowing queuing to and printing on devices, starting and stopping LP from processing requests, changing the configuration of printers and finding the status of the LP system. This memo describes the implementation of LP and suggests how it can be used as a general purpose spooler.

The remainder of this paper is organized as follows: Section 2 presents an overview of the features of LP and defines terms that will be used throughout the memo. See [1] for a detailed description of the role of an LP Administrator. Section 3 tells how to build an LP system. Section 4 describes the LP directory structure and file formats. The internals of the LP scheduler are outlined in Section 5. Section 6 addresses the issue of using LP for general purpose spooling, Section 7 discusses possible extensions to LP and the last section summarizes the features that separate LP from other spooling systems.

2. OVERVIEW OF LP FEATURES

2.1 Definitions

We will define several terms before presenting a brief summary of LP commands. LP was designed with the flexibility to meet the needs of users on different UNIX systems. Changes to LP's configuration (see below) are performed by the *lpadmin*(1M) command. (A parenthesized number immediately following a command name refers to that section of the *UNIX User's Manual.*)

LP makes a distinction between printers and printing devices. A *device* is a physical peripheral device or a file and is represented by a full UNIX path name. A *printer* is a logical name that represents a device. At different points in time, a printer may be associated with different devices. A *class* is a name given to an ordered list of printers. Every class must contain at least one printer. Each printer may be a member of zero or more classes. A *destination* is a printer or a class. One destination may be designated as the *system default destination*. The *lp*(1) command will direct all output to this destination unless the user specifies otherwise. Output that is routed to a printer will be printed only by that printer, whereas output directed to a class will be printed by the first available class member.

Each invocation of *lp* creates an output *request* that consists of the files to be printed and options from the *lp* command line. An *interface program* which formats requests must be supplied for each printer. The LP scheduler, *lpsched*(1M), services requests for all destinations by routing requests to interface programs to do the printing on devices. An LP *configuration* for a system consists of devices, destinations and interface programs.

† UNIX is a trademark of Bell Laboratories.

2.2 Commands

2.2.1 Commands for General Use

Lp(1) is used to request the printing of files. It creates an output request and returns a *request id* of the form:

```
dest−seqno
```

to the user, where *seqno* is a unique sequence number across the entire LP system and *dest* is the destination where the request was routed.

Cancel is used to cancel output requests. The user supplies request ids as returned by *lp* or printer names, in which case the currently printing requests on those printers are canceled.

Disable prevents *lpsched* from routing output requests to printers.

Enable(1) allows *lpsched* to route output requests to printers.

2.2.2 Commands for LP Administrators

Each LP system must designate a person or persons as LP Administrator to perform the restricted functions listed below. Either the super-user or any user who is logged into UNIX as "lp" qualifies as an LP Administrator. All LP files and commands are owned by lp, except for *lpadmin* and *lpsched*, which are owned by root.

Lpadmin(1M) modifies the LP configuration. Many features of this command cannot be used when *lpsched* is running.

Lpsched(1M) routes output requests to interface programs which do the printing on devices.

Lpshut stops *lpsched* from running. All printing activity is halted, but the other LP commands may still be used.

Accept(1M) allows *lp* to accept output requests for destinations.

Reject prevents *lp* from accepting requests for destinations.

Lpmove moves output requests from one destination to another. Whole destinations may be moved at once. This command cannot be used when *lpsched* is running.

3. BUILDING LP

All LP commands are built from source code that resides in the **/usr/src/cmd/lp** directory including the make file, **lp.mk**. All structures and constants that are mentioned below are defined in the header files **lp.h** and **lpsched.h** in the same directory. Unless some of the definitions in **lp.mk** are changed, LP may be installed only by the super-user. Before installing a new LP system, make sure there is a login called **lp** on your system and that the spool directory, **/usr/spool/lp**, does not exist. **Lp**'s login directory may be **/usr/spool/lp** for convenience. To install LP, perform the following:

```
cd /usr/src/cmd/lp
make −f lp.mk install
```

This builds all LP commands and creates the directory structure which is described in the next section. The initial LP configuration produced by the preceding commands consists of no printers, classes or default destination. LP must be configured by an LP Administrator using the *lpadmin* command in order to create a useful spooler.

In addition, add the following code to /etc/rc:

```
rm −f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo "LP scheduler started"
```

This starts the LP scheduler each time that UNIX is restarted.

Several variables in **lp.mk** may be changed before installing LP to customize the system:

Variable	*Default Value*	*Meaning*
SPOOL	/usr/spool/lp	spool directory
ADMIN	lp	logname of LP Administrator
GROUP	bin	group that owns LP commands and data
ADMDIR	/usr/lib	administrator commands reside here
USRDIR	/usr/bin	user commands reside here

If an existing LP spool directory is corrupted (but not the LP programs) or if it needs to be rebuilt from scratch, make sure that *lpsched* is not running and perform the following as super-user:

1. Make copies of any interface programs that are not standard LP software. DO NOT make these copies underneath the spool directory. The path name for printer p is /usr/spool/lp/interface/p.
2. rm −fr /usr/spool/lp
3. make −f lp.mk new

This recreates the bare LP configuration described above.

WARNINGS:

1. Some LP commands invoke other LP commands. Moving them after they are built will cause some commands to fail.
2. The files under the SPOOL directory should be modified *only by LP commands*.
3. All LP commands require set-user-id permission. If this is removed, the commands will fail.

4. DIRECTORY STRUCTURE AND FILE FORMATS

The LP directory structure, as depicted in Figure 1, shows all directories and files that are under the spool directory, **/usr/spool/lp**. Section numbers in Figure 1 refer to the section numbers in this memo in which the appropriate file is described. The notation <**x**> means "zero or more files of type **x**".

4.1 FIFO

FIFO is a fifo (named pipe) special file where all commands send messages to *lpsched*. Any of the LP commands may write to FIFO, but only *lpsched* may read it. A subroutine named **enqueue** sends a message and its arguments to *lpsched* on FIFO. The usage of enqueue is:

```
enqueue(msg, arglist)
char msg;
char *arglist;
```

All messages are defined mnemonically in the LP header file, **lp.h**. *Arglist* is a (possibly null) blank-separated list of arguments associated with the message *msg*. Enqueue returns non-zero if *lpsched* is running and zero if not. Table 1 lists the legal messages to *lpsched*.

File Name	*Section*
spool directory	4.
<lock files>	4.8
<log files>	4.3
FIFO	4.1
class	4.9
<class files>	
default	4.2
interface	4.10
<interface programs>	
member	4.11
<member files>	
model	4.12
<model programs>	
outputq	4.4
pstatus	4.5
qstatus	4.6
request	4.13
<request directories>	
<request files>	
<data files>	
seqfile	4.7

Figure 1. LP Directory Structure

4.2 Default

The **default** file contains the name of the system default destination terminated with a new-line. If this file is absent or empty, the system has no default destination.

4.3 Log Files

The **log** file is a record of *lpsched* errors and printing activity since the time when *lpsched* was last invoked. **Oldlog** contains the same information from the previous invocation of *lpsched*.

The first (last) line of the log indicates the time that *lpsched* was started (stopped). Error messages have the form:

```
lpsched: error-message
```

For each output request that has printed (or is currently printing) there is a line with the following tab-separated fields: request id, logname of requester, printer which serviced the request and the date and time when printing began. There is more than one entry in the log for requests that were restarted after they were partially printed.

4.4 Outputq

The binary file **outputq** is a queue of output request entries that are made by the *lp* command and have the form shown in Figure 2. There is one entry for each pending or partially printed request in addition to the "deleted" entries for output requests that have been serviced since *lpsched* was last invoked. The requests for each printer are serviced strictly on a first in first out basis. **Outputq** entries are marked "deleted" by the *cancel*, *disable* and *lpsched* commands and may be modified by the *lpmove*, *disable* and *lpsched* commands.

TABLE 1. Messages Recognized by *lpsched* on FIFO

MESSAGE	*MEANING TO LPSCHED*
F_ENABLE pr	Printer **pr** has been enabled. Pending requests (if any) will be printed on **pr**.
F_NOOP	No-op to check if *lpsched* is running.
F_DEV pr path	New device for printer **pr** is **path**.
F_STATUS	This causes *lpsched* to dump internal status to the **log** file (see *Log Files*).
F_DISABLE pr	Printer **pr** has been disabled. If it is busy, printing on **pr** will stop. If another printer can service the aborted request, then it will start printing it in its entirety.
F_CANCEL dest seqno	Request id **dest−seqno** has been canceled. If it is currently printing, then printing will stop.
F_NEWLOG	This causes *lpsched* to create a new **log** file (see *Log Files*). The old log file is renamed **oldlog**.
F_REQUEST dest seqno user	Output request id **dest−seqno** has been made by **user**. If there is a printer than can service it, it will be printed immediately.
F_QUIT	This causes *lpsched* to stop running. All printing is terminated.
F_MORE pr	Printer **pr** is ready to print more requests.
F_ZAP pr	Busy printer **pr** has been disabled and its request has been canceled.

```
struct outq {                             /* output queue entry */
      char o_dest[DESTMAX+1];             /* output destination (class or member) */
      char o_logname[LOGMAX+1];           /* logname of requester */
      int o_seqno;                        /* sequence # of request */
      long o_size;                        /* size of request -- # of bytes of data */
      char o_dev[DESTMAX+1];              /* if printing, the name of the printer.
                                             Otherwise, "-". */
      time_t o_date;                      /* date of entry into output queue */
      short o_flags;                      /* See below for flag values */
};

/* Value interpretation for o_flags: */

#define O_DEL         1                   /* Request deleted */
#define O_PRINT       2                   /* Request now printing */
```

Figure 2. Outputq Entry

4.5 Pstatus

The binary file **pstatus** contains one entry of status information for each printer. Printer status entries are detailed in Figure 3. Entries are added and removed by the *lpadmin* command and are modified by the *cancel*, *enable*, *disable* and *lpsched* commands.

```
struct pstat {                          /* printer status entry */
        char p_dest[DESTMAX+1];         /* name of printer */
        int p_pid;                      /* if busy, process id that is printing, otherwise 0 */
        char p_rdest[DESTMAX+1];        /* if busy, the destination designated
                                           by the user to lp, otherwise "-" */
        int p_seqno;                    /* if busy, seq # of printing request */
        time_t p_date;                  /* date last enabled or disabled */
        char p_reason[P_RSIZE];         /* if enabled, then "enabled", otherwise
                                           the reason the printer has been disabled. */
        short p_flags;                  /* See below for flag values */
};

#define P_ENAB      1                   /* printer enabled */
#define P_AUTO      2                   /* disable printer automatically */
#define P_BUSY      4                   /* printer now printing a request */
```

Figure 3. Pstatus Entry

4.6 Qstatus

The binary file **qstatus** contains one entry per destination which tells if the *lp* command is accepting requests. **Qstatus** entries have the form shown in Figure 4 and are added and removed by the *lpadmin* command and modified by the *accept*, *reject* and *lpmove* commands.

```
struct qstat {                          /* queue status entry */
        char q_dest[DESTMAX+1];         /* destination */
        short q_accept;                 /* TRUE iff lp accepting requests for dest,
                                           otherwise FALSE.*/
        time_t q_date;                  /* date status last modified */
        char q_reason[Q_RSIZE];         /* if accepting then "accepting",
                                           otherwise the reason requests for dest are
                                           being rejected by lp */
};
```

Figure 4. Qstatus Entry

4.7 Seqfile

The file **seqfile** contains the sequence number (terminated by a new-line) of the last request id that was assigned by the *lp* command. This number is incremented by *lp* for each request. When it reaches a maximum (defined in **lp.h**) it is reset to 1. If this file is missing then *lp* will create a new file containing the number 1.

4.8 Lock Files

Several lock files are maintained in order to guarantee LP commands exclusive access to data files. They are binary files which contain the process id of the locking process. A list of lock files and their associated data files follows:

Lock File	*Data File*
OUTQLOCK	outputq
PSTATLOCK	pstatus
QSTATLOCK	qstatus
SEQLOCK	seqfile

Lock files "expire" after a given time interval and may be unlinked by any LP process. Thus, commands that lock a data file for longer than this interval must update the modification time on the lock file. The creation, updating and unlinking of lock files is handled automatically by the LP low level file access routines.

Another lock file, **SCHEDLOCK**, is present while *lpsched* is running to ensure that only one invocation of *lpsched* is active. Unlike other lock files, **SCHEDLOCK** has no expiration time.

Caution: any processes that need to concurrently lock more than one lock file should lock them in the following order to avoid deadlock:

OUTQLOCK, PSTATLOCK, QSTATLOCK, SEQLOCK

Failure to release a lock file may also cause deadlock.

4.9 Class

The **class** directory contains one file per LP class which lists the members of the class, one per line. The name of the file is the same as the class name. Each class member is an LP printer and may not be an LP class. Every class must always have at least one member. Class files are created, modified and deleted by the *lpadmin* command.

4.10 Interface

The **interface** directory contains one executable program per printer with the same name as the printer. When *lpsched* chooses an output request, **dest−seqno**, that was requested by user **logname**, to be printed on printer **pr**, it invokes interface program **pr** in the following way:

pr dest−seqno logname title copies options file ...

where
copies is the number of copies requested
title is the optional title supplied to *lp* or null
options is a blank-separated string of options requested by the user to *lp* or null
file is the full path name of a file to be printed

The interface program is invoked with its standard output and standard error output directed to the printer's device. If file access modes permit, the device is opened for reading and writing. The interface's standard input is taken from **/dev/null**. Interface programs may be shell procedures or compiled C programs. They may be supplied by an LP Administrator or selected from a set of model interface programs (see *Model* below). Interface programs are supplied by LP Administrators via the *lpadmin* command.

4.11 Member

The **member** directory contains one file per LP printer with the same name as the printer. The first line of the file is the full path name of the device associated with the printer. Following lines (if any) are the names of classes to which the printer belongs. A printer need not belong to any classes and may belong to more than one. Member files are created, modified and removed by the *lpadmin* command.

4.12 Model

The **model** directory contains several printer interface programs that are distributed with the LP system. The names of these files bear no relationship to LP printers and class names. Copies of these programs may be customized by an LP Administrator to be used as printer interface programs. No new model interfaces can be added to the system.

4.13 Request

The **request** directory contains one directory for each LP destination with the same name as the destination. Each destination's request subdirectory holds information pertaining to pending requests for that destination.

Each request subdirectory contains request files and data files. These files are created by the *lp* command to pass information to *lpsched* and are deleted by the *cancel*, *disable* and *lpsched* commands, and may be moved by the *lpmove* command.

The name of the request file for output request **dest−seqno** is **r−seqno**. It has entries of the form:

    flag value

where *flag* is a single character in column 1, column 2 is blank and an optional *value* starts in column 3. Legal flags, as defined mnemonically in **lp.h**, are summarized in Table 2. The order of entries in a request file is the same order that they are listed in Table 2. The R_TITLE, R_COPIES, R_OPTIONS and one or more R_FILE entries are mandatory.

TABLE 2. Request File Entries

FLAG	*VALUE*
R_TITLE	Optional title supplied to *lp* or null
R_COPIES	Number of copies requested
R_OPTIONS	Printer- and Class-dependent options separated by white space
R_FILE	Name of data file to be printed; any file name not beginning with "/" is assumed to be in the request subdirectory along with the request file
R_MAIL	Logname of person to send mail to after request has been printed
R_WRITE	Logname of person to write to after request has been printed

A request file is associated with zero or more data files. Data files for request **dest−seqno** have the name **dn−seqno**, where **n** is a non-negative integer. These files contain data to be printed.

Examples:

1.
```
$ pr file | lp
request id is x−50 (standard input)
```

   The directory **request/x** will contain the request file **r−50** and the data file **d0−50** which is a copy of the standard input to *lp*. File **r−50**:

```
R_TITLE
R_COPIES 1
R_OPTIONS
R_FILE d0−50
```

2.
```
$ lp -c file1 file2
request id is x-51 (2 files)
```

The −c option causes *lp* to copy files before returning to the user. The directory **request/x** will contain the request file **r−51** and the data files **d0−51** (a copy of file1) and **d1−51** (a copy of file2). File **r−51**:

```
R_TITLE
R_COPIES 1
R_OPTIONS
R_FILE d0-51
R_FILE d1-51
```

3.
```
$ lp file
request id is x-52 (1 file)
```

The directory **request/x** will contain the request file **r−52**. If *file* can be linked to this directory it will be named d0−52. In this case, file **r−52** contains:

```
R_TITLE
R_COPIES 1
R_OPTIONS
R_FILE d0-52
```

On the other hand, if *file* can't be linked, no data file is created and file **r−52** contains:

```
R_TITLE
R_COPIES 1
R_OPTIONS
R_FILE fullfile
```

Fullfile is the full path name of *file*.

5. LP SCHEDULER INTERNALS

5.1 Overview

The LP scheduler, *lpsched*, services requests in the output queue, **outputq**, first come first served, invoking the appropriate interface program to print each request. It is the only demon in the LP system and runs continuously unless it is stopped by the *lpshut* command or the computer system is stopped. It is present even when there are no pending output requests, in which case it sleeps awaiting a message on FIFO.

5.2 Interaction With Other LP Commands

Because it would be inefficient for *lpsched* to perform file I/O each time it needed to know the relationships between printers, classes, requests and devices, *lpsched* maintains its own structures which provide this information more easily. This burdens LP commands by requiring them to inform *lpsched* (on FIFO) of changes to LP data in addition to updating the data files. The former step is required in order to keep the file structure consistent with *lpsched*'s in-memory data. It is this duplication of information that allows LP commands to be used even if *lpsched* is not running.

As an example, let us consider how the *lp* command works. When a request is made to *lp* it builds the request and data files, locks **OUTQLOCK**, adds the new request entry to **outputq**, writes an F_REQUEST message to *lpsched* on FIFO which describes the new request and then unlocks **OUTQLOCK**. The time during which **OUTQLOCK** is locked is a non-interruptible critical section, so signals are ignored. Most LP commands follow this pattern of:

1. lock one or more lock files
2. modify one or more data files
3. send a message to *lpsched* on FIFO
4. unlock the lock files

5.3 Data Structures

When *lpsched* is started it internalizes the information in the LP data files in an in-memory network of circular double-linked lists. Subsequent messages read from FIFO cause this network to be updated so that the lists are kept consistent with the files. The main component of *lpsched*'s lists is the **dest** node shown in Figure 5. There is one of these structures for each destination giving its name and type (class or printer). Nodes that are printers also indicate status (busy or idle, enabled or disabled) as well as information concerning the currently printing request.

```
struct dest {                            /* destination node */
        char *d_dname;                   /* name of destination */
        int d_status;                    /* status of destination -- see below */
        char *d_device;                  /* full path name of device for printer */
        int d_pid;                       /* process id of busy printer */
        struct outlist *d_print;         /* output request currently printing */
        struct dest *d_dnext;            /* next destination */
        struct dest *d_dprev;            /* previous destination */
        struct dest *d_tnext;            /* next destination of same type */
        struct dest *d_tprev;            /* previous destination of same type */
        struct destlist *d_class;        /* class list for printers, member list for classes */
        struct outlist *d_output;        /* list of output requests for dest */
};

/* The following flags are used to interpret dest.d_status */

#define D_PRINTER  1                     /* destination is a printer */
#define D_CLASS    2                     /* destination is a class */
#define D_ENABLED  8                     /* printer is active */
#define D_BUSY     16                    /* printer is busy */
```

Figure 5. Destination Node

Three global **dest** nodes serve as list heads to ease the traversal of the network:

dest	links all destinations in the d_dnext and d_dprev fields
printer	links all printers in the d_tnext and d_tprev fields
class	links all classes in the d_tnext and d_tprev fields

Each printer node contains a linked list of destinations indicating which classes it belongs to. Class nodes have lists of the same format showing which printers are members. Destination lists, as shown in Figure 6, point to **dest** nodes.

```
struct destlist {                       /* destination list node */
        struct dest *dl_dest;           /* pointer to destination */
        struct destlist *dl_next;       /* pointer to next destination */
        struct destlist *dl_prev;       /* pointer to previous destination in list */
};
```

Figure 6. Destination List Node

Because output may be directed to classes or printers, every destination has an associated output request list. Each list is ordered according to the time the F_REQUEST messages were received by *lpsched* from *lp*. The format of output request lists is shown in Figure 7.

```
struct outlist {                        /* output request list node */
        int ol_seqno;                   /* sequence number assigned by lp */
        char *ol_name;                  /* logname of requester */
        int ol_time;                    /* time request was received by lpsched */
        struct dest *ol_dest;           /* pointer to request destination */
        struct dest *ol_print;          /* if printing, a pointer to the printer */
        struct outlist *ol_next;        /* next output request in list */
        struct outlist *ol_prev;        /* previous output request in list */
};
```

Figure 7. Output Request List Node

5.4 Printing a Request

Lpsched is ready to print a request when one of the following messages is received and when one of that message's associated conditions is met:

Message	*Conditions*
F_REQUEST dest seqno user	1. dest is an enabled, idle printer 2. dest is a class which contains an enabled, idle printer
F_MORE pr	1. there is a pending request queued for pr 2. there is a pending request queued for a class which pr belongs to
F_ENABLE pr	1. there is a pending request queued for pr 2. there is a pending request queued for a class which pr belongs to
F_DISABLE pr	pr is busy and the request it is currently printing is queued for a class which contains pr and a member of that class is enabled and idle

When a request is ready to be printed, *lpsched* forks so that its child may do the printing and the parent can continue scheduling other requests. It is the child that executes the interface program and waits for its completion. A non-zero exit status indicates that the interface encountered errors while printing the request. If errors occurred or if the user requested notification of the completion of the printing, mail is sent or a message is written to the requester's terminal. The **outputq** entry is deleted and the request and data files are removed. The parent is informed that the printer is ready to print another request via an F_MORE message on FIFO and the child exits.

Several processes are concurrently active during the printing of a request. Because UNIX systems are typically configured to impose limits on the number of concurrently active processes per user id (except for root) and because most LP programs must be owned by user lp and because they require set-user-id permission, the number of printers that LP can support is affected. Unless the LP system is owned by root (this is not encouraged) there is a limit on the number of LP printers that may be printing simultaneously. The number of active processes per print request includes the child of the scheduler and the interface program and any of its children. Model interface programs, for example, are shell procedures that usually have the form:

```
commands
(
    commands
) |  filter
exit
```

Each request that is printed by a model interface creates two invocations of the shell, an invocation of a device filter and a process to execute a command within the parentheses in addition to a child of the scheduler. With a limit of 25 active processes per user, for example, an LP system with an LP administrator other than root would be able to support up to four line printers using typical model interface programs.

5.5 Cancellation of Requests that are Partially Printed

When a partially printed request is canceled via the *cancel* or *disable* command, the process id found in the **pstatus** entry is signaled with **SIGTERM**. This is the process id of the interface program itself, not the immediate child of the scheduler. It is up to the interface to clean up and then exit. The child of the scheduler waits for the death of the interface and exits. The scheduler then waits for the death of its child.

When *lpsched* is stopped by the *lpshut* command or when it is signaled with **SIGTERM**, it broadcasts this signal to all of its children, which, in turn, terminate the execution of the interface programs. *Lpsched* then removes **SCHEDLOCK** and exits. When printing is terminated this way or by disabling a busy printer (without canceling the request), the requests that were aborted will be reprinted in their entirety.

6. USING LP AS A GENERAL PURPOSE SPOOLER

Although the documentation and commands refer to LP as a line printer spooling system, it was designed with general purpose spooling in mind. Several features allow LP to be customized for a variety of spooling applications. The *lp* command never makes any assumptions about the kind of files it is supplied. No pagination is added and no checks are made for non-ascii input. Thus, *lp* can pass command files, nroff/troff input files, binary data files, executable files, ascii text files, etc. to arbitrary interface programs. *Lp* also allows users to pass options from the *lp* command line to interface programs using the **−o** key letter. It is up to the LP Administrator to supply interface programs to perform the desired functions on input files. The devices associated with printers need not be line printers. They must be writable by LP and may be any type of file (even **/dev/null**). By designing interface programs and by placing new interpretations on destinations and devices LP can perform many diverse functions.

Example:

Many installations use special purpose software to batch *nroff* requests so that they can limit the number of concurrently executing *nroff* commands. LP can be used for batch processing of this and other commands that place a heavy load on the system. Each "printer" can be thought of as a command processor. Input files (built by a front-end interface to *lp*) are shell procedures which contain *nroff* command lines and environment information. The role of the interface program is to execute the nroff command in the user's environment as of the time *lp* was invoked. The output may be directed to a file or a printer designated by the user. The devices associated with the processors could be log files or line printers. By grouping *n* of these processors in a class, users are limited to *n* concurrent executions of frequently used, heavy load commands. Furthermore, queuing to these destinations and the running of the command processors are under the control of the *accept*, *reject*, *enable* and *disable* commands.

7. EXTENSIONS

It is hoped that any future enhancements to LP will not take away from its generality. It would have been easy, for instance, to add dozens of printer-specific options to the *lp* command. This was not done because LP makes it easy for LP Administrators to add these options in printer interface programs and for users to take advantage of them by sliding them past *lp*. On the other hand, there are enhancements that could make LP even more useful while retaining its generality.

Lpsched services requests on a first come first served basis. This may be undesirable when there are a limited number of printers and it is desirable to schedule small print jobs ahead of larger ones. Care must be taken to avoid penalizing the larger requests too severely. *Lpsched* could be enhanced to enforce such a scheduling discipline. User-assigned priorities could be added to the *lp* command in order to affect *lpsched*'s scheduling algorithm. Another useful feature is to allow users to inhibit requests from printing while leaving them queued. Subsequently, the held requests could be released or canceled. Another enhancement to LP would allow different queues to build for the same destination in order to implement the idea of "peak period" or "overnight" queues.

The *lpadmin* command requires that *lpsched* must not be running before it is going to attempt to alter the LP configuration. This restriction was imposed to simplify the initial version of *lpsched*. In cases where a configuration is frequently undergoing changes it is a nuisance to have to shut the scheduler before using *lpadmin*. Shutting the scheduler, of course, means that all printing stops.

The above features were not considered absolutely essential and would have greatly increased the complexity of the initial version of the package. The author believes that it would not require a major effort to add these new capabilities to LP. The design would not need to be radically changed to introduce these enhancements.

8. SUMMARY

To the best of the author's knowledge, LP is the only centrally supported spooler under UNIX which offers all of the following features in a single package:

- Printers may be grouped into classes.
- Each printer may belong to several or no classes.
- The spooler may be reconfigured to meet the needs of specific users.
- The spooling function is separated from the printing function. Any device or writable file may be spooled to by a user-supplied interface program.
- LP can be used for off-line printing as well as for other spooling functions.

REFERENCE

[1] Kliegman, J. R. *LP Administrator's Guide*, Bell Laboratories.