

AIX job spooler

Process and Resource Control

Skill Level: Introductory

[Mr. Dana L. French \(dfrench@mtxia.com\)](mailto:dfrench@mtxia.com)

President
Mt Xia Inc.

28 Apr 2009

Job spooling or spooled processing are mainframe concepts that are available in the IBM® AIX® environment, but are rarely implemented. This article describes the configuration, reasons, and purposes for implementing job spooling in AIX environments.

Introduction

The IBM AIX job spooler is a very useful built-in utility on all AIX systems but is rarely enabled or utilized. In this article, the term "job" is used to represent any application, utility, program, or shell script that can be executed on an AIX system, but does not require interactive responses from the user or administrator. The AIX job spooler is a utility that provides a mechanism for administrators and users to submit multiple jobs and then process each job one at a time. These jobs are processed in the sequence in which they were submitted to the spooler and is a useful system administration tool from a variety of perspectives. For example, some of the purposes of job spooling include:

- Reducing workload of systems administrators
- Processes that require single threading
 - Automated editing of data files
 - Accessing data from dynamic files

- Input or output to/from attached devices
- Limiting the number of concurrent processes
- Controlling system load
- Job control
- Job scheduling and tracking
- Reserving resources for on-line processing during regular business hours
- Batch processing during off-hours

In the modern data center environment, the need for a job spooler is greater than ever. Much of the emphasis on managing large environments revolves around data center automation, which requires the ability to schedule processes and to be confident in the results of that processing. For example, automated system deployment requires that storage devices, network addresses, and virtual I/O to be dynamically allocated during the build out of virtualized systems. Since automated deployment requests may be initiated by multiple sources, a mechanism must be employed to ensure that physical and virtual resources assigned to a system are unique. This is where job spooling can provide a solution. System deployment requests can be generated by multiple sources and accepted (queued) by the job spooler. These queued requests are then processed one at a time by the job spooler. Single threading the requests through the job spooler eliminates the problem of duplicating physical or virtual resource assignments that can occur if multiple requests are processed simultaneously.

AIX job spooler configuration

The AIX job spooler is integrated into the queue daemon, which is the same queue daemon that controls the printers. Normally the job spooler queue bsh is disabled by default. Enabling the bsh queue is a simple process that requires editing the queue configuration file `/etc/qconfig`. First, view the `/etc/qconfig` file and observe the queue definition stanza labeled "bsh:" and its associated queue device labeled bshdev:". These stanzas are commented out of the default configuration by an asterisk "*" at the beginning of each line:

```
# view /etc/qconfig
...
...
*
* BATCH queue for running shell scripts
*
*bsh:
* device = bshdev
* discipline = fcfs
*bshdev:
```

```
* backend = /usr/bin/bsh
```

As the user root or as a user that is a member of the printq group, edit the /etc/qconfig file and remove the asterisks from the beginning of the lines for the "bsh:" and "bshdev:" stanzas:

```
# vi /etc/qconfig
...
...
...
*
* BATCH queue for running shell scripts
*
bsh:
    device = bshdev
    discipline = fcfs
bshdev:
    backend = /usr/bin/bsh
```

After modifying the queue daemon configuration file /etc/qconfig, stop and restart the queue daemon using the System Resource Controller commands:

```
# lssrc -a | grep qdaemon
# stopsrc -s qdaemon
# startsrc -s qdaemon
# lssrc -a | grep qdaemon
```

After stopping and restarting the queue daemon, check the queue status using the lpstat command:

```
# lpstat -W
Queue          Device          Status
----          -
bsh            bshdev          READY
```

AIX job spooler implementation

A simple example of using the AIX job spooler for executing batch jobs is shown below. This example uses the "at" scheduler to submit a job to the spooler at a specific date and time. The date and time in the example is "now," but can be changed to any date and time desired. The first portion of the example creates a shell script that outputs some information to a file in the /tmp directory. This script also sleeps for a random number of seconds from 0 to 99; the purpose of the sleep is to provide some delay time in order to permit the user to view the contents of the queue. Create the example shell script by executing the following commands:

```
print -- '#!/usr/bin/ksh93
```

```
typeset -R2 T
T=${RANDOM}
print -- "${T} ${0}" > "${0}.out"
date >> "${0}.out"
sleep ${T}
date >> "${0}.out"
' > /tmp/tmp1.ksh
```

After creating the /tmp/tmp1.ksh shell script, change the permissions on the file to make it executable.

```
chmod 755 /tmp/tmp1.ksh
```

Submitting a job to the job spooler queue and to the at job scheduler simultaneously requires a compound command. The example command to submit the shell script to the job spooler is `lp -d bsh /tmp/tmp1.ksh`. To schedule the spooler command, send it to the standard input of the `at` command using a Korn shell print as shown:

```
print -- "lp -d bsh /tmp/tmp1.ksh" | at now
```

The `at` scheduler will execute the command immediately if the date/time of "now" was specified. Use the `lpstat -t` command to view the contents of the job spool queue:

```
# lpstat -t
Queue      Dev      Status   Job      Name
Submitted          Rnk  Pri      Blks  To
-----
bsh        bshde  RUNNING   86      /tmp/tmp1.ksh
03/21/09  20:03:43   1  15      1    1
                /tmp/tmp1.ksh
                PP %
                ---
                0  0
```

When the job is complete, a file named /tmp/tmp1.ksh.out will be created with contents similar to the following:

```
# cat /tmp/tmp1.ksh.out
50 /tmp/tmp1.ksh
Sat Mar 21 20:03:43 CDT 2009
Sat Mar 21 20:04:33 CDT 2009
```

The first line of the output contains the random number of seconds selected for the sleep time, followed by the name of the script. The next two lines contain a date/time stamp from before and after the sleep command. The difference between the time stamps should be the number of seconds from the first line.

By using the above job spooling concepts, the systems administrator can be relieved

of the duty of determining the load placed on the system by each overnight batch job. Normally the system administrator attempts to spread out the workload on a system by scheduling job processing through cron. The administrator attempts to schedule job processes at incremental time slots based on expected duration of each job execution. Some additional time is usually added to the scheduled time of the next job, in case the previous job runs longer than expected.

Using the AIX job spooler, all overnight non-sequential batch jobs can be kicked off at the same time because the spooler runs each job one at a time in the order they were submitted to the spooler. So, even if 50 jobs are scheduled to run at 23:00, since they are spooled, they run sequentially one at a time. This eliminates the need for the administrators to spread out the jobs so the system is not overloaded. They can schedule the jobs to run at any time, as long as they send them to the job spooler, which will single thread each job through the spooler. The administrators are no longer burdened with determining system load and scheduling times for available system resources.

cron scheduling

The same technique shown previously using the `at` scheduler can of course be used with the cron scheduler as well. The command to execute in the crontab record would simply be preceded by the spooler command. An example crontab record line that runs a command at 23:00 PM every night would appear as:

```
0 23 * * * /usr/bin/lp -d bsh /usr/local/scripts/schedjob.ksh
```

Again, when scheduling non-sequential batch jobs, they can all be scheduled at 23:00 because they will be queued one at a time through the job spooler. No effort is required by the administrator to evaluate system load or to arrange the processing schedule.

Back-up processing

Back-up processing jobs should be scheduled some time after all overnight processing jobs so they are the last job in the spooler queue. If the last overnight processing job is scheduled to be submitted to the job spooler at 23:45, then back-up processing should be submitted after that time, as an example:

```
0 0 * * * /usr/bin/lp -d bsh /usr/local/scripts/backupjob.ksh
```

Even if the overnight processing does not complete until 05:00 AM, since the back-up job is the last job in the queue, it does not start until the overnight processing is complete.

Sequential processing

Jobs that require sequential processing can be submitted to the job spooler through cron by running each job one minute apart. Since the jobs are queued on the spooler, as soon as one job completes, the next job starts. The administrator no longer has to guess at timing between jobs, or create temporary files to trigger processing of the next job. Job spooling ensures that each job is processed in sequence and is independent of variations in processing time. An example crontab records representing jobs that require a specific processing sequence is shown as follows:

```
0 23 * * * /usr/bin/lp -d bsh /usr/local/scripts/schedjob01.ksh
1 23 * * * /usr/bin/lp -d bsh /usr/local/scripts/schedjob02.ksh
2 23 * * * /usr/bin/lp -d bsh /usr/local/scripts/schedjob03.ksh
3 23 * * * /usr/bin/lp -d bsh /usr/local/scripts/schedjob04.ksh
```

Multiple job spools

If system resources are available, additional job spools can be created to allow multiple jobs to be processed simultaneously through the spooler.

```
# mkquedev -q bsh -d bshdev2 -a 'backend = /usr/bin/ksh93'
```

Notice the Korn Shell 93 `/usr/bin/ksh93` command shell is specified with the `mkquedev` command; any valid shell can be used as the back-end processor. As many queue devices can be added to the job queue as needed to efficiently utilize the system resources, while maintaining a controlled spooling environment. After adding another queue device, an `lpstat -W` shows multiple queue devices associated with the `bsh` job queue. The `bsh` job queue now allows a spooled job to be processed on each available device; if two devices are available, then two jobs can be processed simultaneously, one on each device.

```
# lpstat -W
Queue      Device      Status
----      -
bsh        bshdev      READY
bsh        bshdev2     READY
```

Recognize that if you use multiple job spools and spool your back-up job, you cannot be assured that all other jobs will be complete before the back-up job is started.

To create a shell script to use as an example for demonstrating job spooling across multiple job queue devices, run the following commands:

```
print -- '#!/usr/bin/ksh93'
```

```
typeset -R2 T
T=${RANDOM}
print -- "${T} ${0}" > "${0}.out"
date >> "${0}.out"
sleep ${T}
date >> "${0}.out"
' > /tmp/tmp1.ksh
chmod 755 /tmp/tmp1.ksh
```

To view an example of job spooling across multiple queues (assuming you created the second job spool queue device), run the following "for" loop to submit ten (10) jobs to the "bsh" job queue.

```
for i in 0 1 2 3 4 5 6 7 8 9
do
    lp -d bsh /tmp/tmp1.ksh
done
```

Now, use the following "while" loop to view the job queue status of the previously submitted jobs. It will sleep for five seconds between each execution of the lpstat -W command:

```
while true
do
    lpstat -W
    print -- "# Hit Control-C to end this loop"
    sleep 5
done
```

When finished viewing the output from the lpstat -W command, type a **Control-C** to interrupt the "while" loop. The output from the "while" loop appears similar to the following. Notice that multiple jobs are in the RUNNING status and as each job completes, a QUEUED job is moved to the job spool device queue and executed.

Queue	Dev	Status	Job Submitted	Name	From Rnk Pri	To Blks Cp	PP %
bsh	bshde	RUNNING	76	/tmp/tmp1.ksh	dfrench 1 15	dfrench 1 1	0 0
bsh	bshde	RUNNING	77	/tmp/tmp1.ksh	dfrench 2 15	dfrench 1 1	0 0
		QUEUED	78	/tmp/tmp1.ksh	dfrench 3 15	dfrench 1 1	
		QUEUED	79	/tmp/tmp1.ksh	dfrench 4 15	dfrench 1 1	
		QUEUED	80	/tmp/tmp1.ksh	dfrench 5 15	dfrench 1 1	
		QUEUED	81	/tmp/tmp1.ksh	dfrench 6 15	dfrench 1 1	

```

QUEUED      82      /tmp/tmp1.ksh  dfrench      dfrench
03/21/09 18:16:14   7 15          1 1
      /tmp/tmp1.ksh
QUEUED      83      /tmp/tmp1.ksh  dfrench      dfrench
03/21/09 18:16:14   8 15          1 1
      /tmp/tmp1.ksh
QUEUED      84      /tmp/tmp1.ksh  dfrench      dfrench
03/21/09 18:16:14   9 15          1 1
      /tmp/tmp1.ksh
QUEUED      85      /tmp/tmp1.ksh  dfrench      dfrench
03/21/09 18:16:14  10 15         1 1
      /tmp/tmp1.ksh

```

Spooling sequential job processes on a system with multiple job queues requires that all jobs be directed to a single job queue device. Otherwise, the jobs would be sent to the first available job queue device. Using the previous for loop example, it can be observed that execution of the queued jobs can be restricted to a single job queue device (bshdev2):

```

for i in 0 1 2 3 4 5 6 7 8 9
do
    lp -d bsh:bshdev2 /tmp/tmp1.ksh
done

while true
do
    lpstat -W
    print -- "# Hit Control-C to end this loop"
    sleep 5
done

```

Restricting the execution of the queued jobs only to the bshdev2 job queue device results in those jobs being executed sequentially. Without specifying bshdev2, each job would be executed on the first available queue device, which would result in multiple jobs running simultaneously (non-sequentially).

If you wish to remove the second job spooler queue device so that only one job at a time is executed, run the following commands:

List the queue device and verify it is the correct one to remove:

```

# lsquedev -q bsh -d bshdev2
bshdev2:
    backend = /usr/bin/ksh93

```

Remove the queue device:

```

# rmquedev -q bsh -d bshdev2

```

Attempt to list the queue device again. It should return an error since it was removed by the previous command:

```
# lsquedev -q bsh -d bshdev2
lsquedev: (FATAL ERROR): 0781-190 Queue:device, bsh:/bshdev2: not found in qconfig
file. Not printed.
```

List the default queue device to verify it still exists:

```
# lsquedev -q bsh -d bshdev
bshdev:
    backend = /usr/bin/ksh93
```

Conclusions

The built-in job spooler provides a solution for many problems and requirements in an AIX environment. Only a few purposes for utilizing the AIX job spooler have been dealt with in detail here, but certainly many others exist. Some examples include:

- Reducing the management workload of the systems administrator.
- Normalizing the utilization of system resources by eliminating peaks and valleys associated with overloaded or underloaded systems.
- Increasing the reliability of cron scheduled job processing by eliminating errors caused by variations in processing time.
- Programmatically obtaining enterprise-wide unique identifiers, such as IP addresses, from a database or flat file.
- Programmatically testing a value for enterprise-wide uniqueness, such as a host name.
- Programmatically reserving physical or virtual resources, such as virtual I/O adapters.
- Eliminating data file update problems associated with simultaneous access from multiple sources.

If you choose to enable the AIX job spooler, be aware that any user on the system will be able to submit processes to the job spooler. Those processes will be executed as the user who submitted the job, and will have the same permissions and settings as that user. So jobs submitted by root will have root-level permissions.

When building shell scripts to submit to the job spooler, it is usually a good idea to define the PATH environment variable inside the script. This is to ensure that all programs, utilities, functions, scripts, and more called by the script are accessible using a listed directory in the variable.

Multiple job queue devices can be created to increase the number of simultaneous

jobs processed, and thus increase the system load and throughput. However, if multiple job queue devices are configured and enabled, those processes requiring sequential execution must be restricted to a single job queue device.

Job control can be achieved by using the normal queue management commands "disable" and "enable." These commands can be used to stop and start job queue devices thus stopping and starting job processing.

Resources

Learn

- AIX Commands: [mkque](#), [mkquedev](#), [lsque](#), [lsquedev](#), [rmque](#), [rmquedev](#), [lpstat](#), [enable](#), [disable](#)
- [Enterprise Wide Unique identifier generator](#) generates an Enterprise Wide Unique UID number for any given user name.
- [New to AIX and UNIX?](#): Visit the "New to AIX and UNIX" page to learn more about AIX and UNIX.
- [AIX Wiki](#): A collaborative environment for technical information related to AIX.
- [The AIX and UNIX developerWorks zone](#) provides a wealth of information relating to all aspects of AIX systems administration.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Discuss

- Participate in the AIX and UNIX forums:
 - [AIX Forum](#)
 - [AIX Forum for developers](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant Forum](#)
 - [Performance Tools Forum](#)
 - [Virtualization Forum](#)
 - More [AIX and UNIX Forums](#)

About the author

Mr. Dana L. French

Mr. French's career in the IT industry has spanned three decades and numerous industries. His work focuses primarily on the fields of business continuity, disaster recovery, and high availability, and he has designed and written numerous software packages to automate the processes of business continuity, disaster recovery and high availability. He is most noted for his approach to system administration as an

automated, business oriented process, rather than as a system oriented, interactive process. He is also a noted authority on the subject of Korn Shell programming.