Chapter 5

CPU: Monitoring

AIX systems administrators have much more at their disposal than the average Unix administrator. Not only can you use the standard Unix generic monitoring tools that have been around nearly as long as Unix itself, but a potpourri of AIX-specific commands is also available. Some of these commands come standard with an AIX build, while others are tools that, although not officially supported by IBM, are widely distributed and are used by most administrators. We'll discuss all these types of monitoring tools in this chapter, including those we don't use very often.

As we go through the tools, note that four commands — **mpstat**, **sar**, **topas**, and **vmstat** — have been enhanced in AIX 5.3 to enable the tools to report back accurate statistics about shared partitions using Advanced Power Virtualization (PowerVM). The trace-based tools **curt**, **filemon**, **netpmon**, **pprof**, and **splat** have also been updated. One command not covered here, **lparmon**, is the most comprehensive tool you can use in a partitioned environment.

vmstat (Unix-generic)

```
vmstat [-fsviItlw] [[-p|-P] pagesize|ALL] [Drives] [Interval [Count]]
```

While the **vmstat** command is more commonly associated with viewing information about virtual memory (hence the "vm"), it is the first tool most administrators invoke when trying to get a quick assessment of their systems. That's because **vmstat** reports back all kinds of pertinent performance-related information, including data about memory, paging, blocked I/O, and overall CPU activity. Because it reports virtually all subsystem information line by line in a quick and painless way, running **vmstat** is probably the simplest and most efficient way to gauge exactly what is going on in your system.

A common way to run **vmstat** is for five iterations every two seconds:

```
vmstat 2 5
```

Running the command in this way produces the following results:

```
# vmstat 2 5

System configuration: lcpu=4 mem=3072MB ent=0.40

kthr    memory              page              faults              cpu
----- ------------- ---------------------  ---------- ----------------------
 r  b    avm    fre    re  pi   po  fr   sr   cy   in  sy   cs  us  sy  id  wa    pc    ec
 1  0 128826 641397  0   0    0   0    0    0  448  87  138   0   1  98   0  0.01   2.8
 1  0 128826 641397  0   0    0   0    0    0  385  10  136   0   1  99   0  0.01   2.2
 1  0 128826 641397  0   0    0   0    0    0  381  13  138   0   1  99   0  0.01   2.2
 1  0 128826 641397  0   0    0   0    0    0  364  40  138   0   1  99   0  0.01   2.4
 1  0 128826 641397  0   0    0   0    0    0  610  13  138   0   2  98   0  0.01   3.3
```

In addition to specific monitoring information, **vmstat** provides a very high-level snapshot of the system, which can be useful. Just by running **vmstat** in the preceding snapshot, we know that we have a system with four logical CPUs and 3 GB of RAM and are using shared processors. (In actuality, this partition is using two physical CPUs; symmetric multithreading is enabled, yielding the four logical CPUs. More about SMT later.)

Some of the more important fields in the **vmstat** output include the following:

- *r* — The average number of runnable kernel threads over the sampling interval you have chosen.

- *b* — The average number of kernel threads in the virtual memory waiting queue over the sampling interval. The **r** value should always be higher than **b**; if it is not, you probably have a CPU bottleneck.
- *fre* — The size of the memory free list. Don't worry too much if this number is really small. More important, determine whether any paging is going on if this size is small.
- *pi* — Pages paged in from paging space.
- *po* — Pages paged out to paging space.

Our focus in this chapter is on the last section of output, CPU:

- *us* — User time
- *sy* — System time
- *id* — Idle time
- *wa* — Time spent waiting on I/O
- *pc* — Number of physical processors consumed (displayed only if the partition is configured with shared processors)
- *ec* — Percentage of entitled capacity (displayed only if the partition is configured with shared processors)

Clearly, the system in our example has no bottleneck to speak of. How can we tell this? Let's look at *us* and *sy*. If these entries combined consistently averaged more than 80 percent, you more than likely would have a CPU bottleneck. If you are in a state where the CPU is running at 100 percent (which happens on occasion to everyone), your system is really breathing hot and heavy. If the numbers are small but the wait time (*wa)* is on the high side (usually greater than 30), this usually signals that there may be I/O problems, which in turn can cause the CPU not to work as hard as it can. Alternatively, if more time is spent in *sy* time than *us* time, your system is probably spending less time crunching numbers and more time processing kernel data. When this happens, it is usually a sign either of badly written code or that something has run amok.

Let's look at another system:

```
# vmstat 2 5

System configuration: lcpu=4 mem=3072MB ent=0.40

kthr    memory              page              faults              cpu
----- ------------ --------------------- ------------ ----------------------
 r  b   avm   fre   re  pi  po  fr  sr  cy  in    sy  cs  us sy id wa    pc    ec
 2  1 169829 600290  0   0   0   0   0   0 553 36538 175 64 32  4  0  0.79  84.9
 3  2 169829 600290  0   0   0   0   0   0 778 33033 175 60 29 11  0  0.84  73.2
 4  1 169828 600291  0   0   0   0   0   0 403 11904 179 76 10  4 10  0.69  87.8
 2  1 169828 600291  0   0   0   0   0   0 368 30745 175 82 14  2  2  0.91  85.5
 6  2 169830 600289  0   0   0   0   0   0 395 27898 173 57 34  4  5  0.89  91.5
```

What kind of determination can we make here? When we add *us* and *sy*, our numbers come out much differently this time — fairly close to 100 percent. This system is clearly CPU-bound. If paging were going on, we would see numbers in the paging (*page*) columns. In this case, no paging is occurring, nor are there any I/O problems to speak of. Because **vmstat** is an all-purpose utility, it can help you perform this quick-and-dirty analysis on the fly. If the blocked processes represented three times the number of runnable processes and everything else stayed the same, I/O would likely be causing the CPU bottleneck. In that case, you should be prepared to have even more of a CPU bottleneck once you fix the I/O problem. As I explained previously, this is all part of systems tuning; fixing one bottleneck often causes another.

sar (Unix-generic)

```
sar {-A [-M]|[-a][-b][-c][-d][-k][-m][-q][-r][-u][-v][-w][-y][-M]}
   [-s hh[:mm[:ss]]] [-e hh[:mm[:ss]]]
   [-P processor_id[,...] | ALL]
   [-f file] [-i seconds] [-o file] [interval [number]]
   [-X file] [-i seconds] [-o file] [interval [number]]
```

The **sar** command is the Unix System Activity Reporting tool (part of the **bos.acct** fileset). It is most commonly used to analyze CPU activity. The command writes to standard output the contents of the cumulative activity, similar to **vmstat**. The default version of **sar** produces a CPU utilization report:

```
# sar 2 5

AIX lpar30p682e_pub 3 5 00CED82E4C00    12/24/07
System configuration: lcpu=4 ent=0.40 mode=Uncapped

10:13:40    %usr    %sys    %wio   %idle   physc   %entc
10:13:42      13      31       0      57    0.18    44.5
10:13:44      12      30       0      58    0.17    43.5
10:13:46      14      35       0      51    0.20    50.8
10:13:48       6      11       0      83    0.07    18.0
10:13:50       9      24       0      67    0.14    34.5

Average       11      26       0      63    0.15    38.3
```

Used this way, the **sar** command provides the same type of high-level information that **vmstat** does, although it also lets you know the mode in which the system is running, which is helpful. In the example, we can see that our partition is an uncapped partition, which, when configured as such, lets the partition use more resources than its entitled capacity. In this default view, the fields themselves are the same as the **vmstat** fields, but *us* becomes *usr*, *sy* becomes *sys*, *id* becomes *idle*, *io* becomes *wio*, *pc* becomes *physc*, and *ec* becomes *entc*.

A more effective way to run **sar** is to view all processors by using the **ALL** flag:

```
# sar -u -P ALL 2 5

AIX lpar30p682e_pub 3 5 00CED82E4C00    12/24/07
System configuration: lcpu=4 ent=0.40 mode=Uncapped

10:24:18 cpu    %usr    %sys    %wio   %idle   physc   %entc
10:24:20  0       27      71       0       2    0.15    37.5
          1        0      35       0      65    0.00     0.5
          2        0      36       0      64    0.00     0.0
          3        0      29       0      71    0.00     0.0
          U        -       -       0      62    0.25    61.8
          -       10      27       0      63    0.15    38.2
10:24:22  0       32      66       0       2    0.15    37.2
          1        0      37       0      63    0.00     0.6
          2        0      35       0      65    0.00     0.0
          3        0       30       0       70    0.00      0.0
          1        0       37       0       63    0.00      0.6
          2        0       35       0       65    0.00      0.0
          3        0       30       0       70    0.00      0.0
          U        -        -       0       62    0.25     62.1
          -       12       25       0       63    0.15     37.9
10:24:24  0       29       69       0        2    0.15     37.7
```

I prefer using **vmstat** to **sar** because **vmstat** provides a quick snapshot of all subsystems, not just CPU. Although you can use other flags to obtain additional subsystem information using **sar**, it just is not as efficient or simple.

One advantage **sar** provides that **vmstat** does not is the ability to capture information and analyze data. This is done through the System Activity Data Collector (**sadc**), which is essentially a back end to **sar**. When enabled through **cron** (it is commented out on a typical default AIX partition), **sadc** collects data periodically in binary format. In the following example, we run it from the command line:

```
# /usr/lib/sa/sadc 2 5 /tmp/sarinfo
```

To view the results (remember it's in binary format), we need to use the **–f** flag:

```
# sar -f /tmp/sarinfo

AIX lpar30p682e_pub 3 5 00CED82E4C00    12/24/07
System configuration: lcpu=4 ent=0.40 mode=Uncapped

10:41:42    %usr    %sys    %wio   %idle   physc   %entc
10:41:44       0       1       0      99    0.01     2.4
10:41:46       0       1       0      98    0.01     2.6
10:41:48       0       1       0      99    0.01     2.1
10:41:50       0       1       0      99    0.01     1.9
Average        0       1       0      99    0.01     2.3
```

iostat (Unix-generic)

```
iostat [-a][-l][-s][-t][-T][-z] [{-A [-P] [-q|Q]} | {-d|-D [-R]} ]
   [-m] [Drives] [Interval [Count]]
```

The **iostat** command is another good first-impression type of tool, which is more commonly used for I/O information. When run with the **-t** flag, it provides only *tty/cpu* information. I also like to use the **-T** flag to obtain the timestamp:

```
# iostat -tT 1

System configuration: lcpu=4 ent=0.40

tty:   tin    tout   avg-cpu: % user % sys % idle % iowait physc % entc  time
       0.0    41.0               0.0   1.1   98.8      0.0   0.0    2.2  10:51:13
       0.0   182.0               0.0   0.9   99.0      0.0   0.0    1.8  10:51:14
       0.0    92.0               0.0   0.9   99.1      0.0   0.0    1.7  10:51:15
       0.0    92.0               0.1   1.1   98.8      0.0   0.0    2.1  10:51:16
       0.0    92.0               0.0   1.4   98.6      0.0   0.0    2.7  10:51:17
```

w (Unix-generic)

```
/usr/bin/w64 [ -hlsuwX ] [ user ]
```

The **w** command prints a summary of all current activity on the system. I like this command — always have and always will. Sometimes I run it even before **vmstat**. I appreciate the clear, concise way in which **w** provides important information, such as load average. You can tell a lot about your system from the load average. If my load average commonly varies between 2 and 5 but is 37 when I run this command, I'm about ready to say, "Houston we have a problem." In the following case, we're okay.

```
# w

  08:29AM   up 1 day,  23:44,  2 users,  load average: 1.00, 1.00, 1.01

User     tty          login@       idle      JCPU      PCPU what
u0004773 pts/0       06:40AM          0         0         0 -ks
u0004773 pts/1       08:28AM          0         0         0 -ksh
```

lparstat (AIX-specific)

```
lparstat { -i | [-H|-h] [Interval [Count]] }
```

The purpose of the **lparstat** command is to report logical partition (LPAR) information statistics. This command also displays hypervisor statistical data about many POWER Hypervisor calls. Introduced in AIX 5.2, **lparstat** is commonly used to assist in shared-processor partitioned environments.

In the following command output, you should recognize the entries up until entitled capacity (*entc*).

```
# lparstat 2 5

System configuration:
  type=Shared mode=Uncapped smt=On lcpu=4 mem=3072 psize=16 ent=0.40

%user  %sys  %wait  %idle physc %entc  lbusy  vcsw phint
-----  ----  -----  ----- ----- ----- ------  ---- -----
  0.1   1.4    0.0   98.5  0.01   2.6    0.0   582     0
  0.0   1.4    0.0   98.6  0.01   2.6    0.0   635     0
  0.0   1.3    0.0   98.7  0.01   2.4    0.0   593     0
  0.0   1.5    0.0   98.5  0.01   2.8    1.2   685     0
  0.1   1.1    0.0   98.8  0.01   2.1    0.0   458     1
```

On shared partitions, **lparstat** provides the following information:

- *lbusy* — The percentage of logical processor utilization (executing at the user and system level)
- *vcsw* — The number of virtual context switches that are virtual processor hardware preemptions
- *phint* — The number of phantom interrupts (redirected to other partitions in the shared pool)

An important flag worth a mention is the **-h** flag, which shows the POWER Hypervisor statistics:

```
# lparstat -H 2 5

System configuration:
  type=Shared mode=Uncapped smt=On lcpu=4 mem=3072 psize=16 ent=0.40

           Detailed information on Hypervisor Call

Hypervisor   Number of   %Total Time   %Hypervisor   Avg Call   Max Call
  Call         Calls        Spent       Time Spent   Time(ns)   Time(ns)

remove         0           0.0           0.0           1          656
read           0           0.0           0.0           1            0
nclear_mod     0           0.0           0.0           1            0
page_init    265           0.0           0.9         604         6593
clear_ref      0           0.0           0.0           1            0
protect        0           0.0           0.0           1            0
put_tce        0           0.0           0.0           1            0
xirr         565           0.1           2.4         758         1406
```

Hypervisor information includes:

- *Number of calls* — The number of Hypervisor calls
- *%Total Time Spent* — Percentage of total time spent on call
- *%Hypervisor Time Spent* — Percentage of Hypervisor time spent on call
- *Avg Call Time* — Average call time for this type of call; the percentage of logical processor utilization executing at the user and system level (in nanoseconds)
- *Max Call Time* — Maximum call time for this type of call (in nanoseconds)

For partitions running AIX 5.2 or AIX 5.3, either in a dedicated environment or in shared and capped mode, the overall CPU utilization is based on the *user*, *sys*, *wait*, and *idle* values. In AIX 5.3 partitions running in uncapped mode, the utilization is based on the entitled capacity percentage.

mpstat (AIX-specific)

```
mpstat [ { -a | -d | -i | -s | -h } ] [ -w ] [ interval [ count ] ]
```

The **mpstat** command (part of the **bos.acct** fileset) was introduced in AIX 5.3. This tool displays overall performance numbers for all logical CPUs on your partitioned system. When you run the command, two sections of statistics are displayed. The first section shows system configuration information, which is displayed when the command starts and whenever a change in the system configuration occurs; the second section, which is displayed at user-specified intervals, shows utilization statistics:

```
# mpstat 1 2

System configuration: lcpu=4 ent=0.4 mode=Uncapped

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0   18    0    0  524  125   56    1    0 100  100  8 58  0 34 0.01  2.1  465
  1    0    0    0  108    0    0    0    0   -    0  0 36  0 64 0.00  0.5  108
  2    0    0    0   10    0    0    0    0   -    0  0 32  0 68 0.00  0.0   10
  3    0    0    0   10    0    0    0    0   -    0  0 29  0 71 0.00  0.0   10
  U    -    -    -    -    -    -    -    -   -    -  -  -  0 97 0.39 97.3    -
ALL   18    0    0  652  125   56    1    0 100  100  0  1  0 98 0.01  2.7  593
--------------------------------------------------------------------------------


  0    3    0    0  392  127   58    1    0 100   67  5 56  0 38 0.01  1.4  331
  1    0    0    0   70    0    0    0    0   -    0  0 34  0 66 0.00  0.4   70
  2    0    0    0   10    0    0    0    0   -    0  0 32  0 68 0.00  0.0   10
  3    0    0    0   10    0    0    0    0   -    0  0 29  0 71 0.00  0.0   10
  U    -    -    -    -    -    -    -    -   -    -  -  -  0 98 0.39 98.2    -
ALL    3    0    0  482  127   58    1    0 100   67  0  1  0 99 0.01  1.8  421
```

Information given includes:

- *cpu* — Logical CPU processor ID
- *min* — Minor page faults
- *ma* — Major page faults
- *mpc* — Total number of interprocessor calls
- *int* — Total number of interrupts
- *cs* — Total number of voluntary context switches
- *ics* — Total number of involuntary context switches

- *rq* — Total run queues
- *mig* — Total number of thread migrations
- *lpa* — Logical processor affinity
- *sysc* — Total number of system calls
- *us* — CPU time spent on user activity
- *sy* — CPU time spent on system activity
- *wa* — CPU time spent waiting on I/O
- *id* — CPU time idle
- *pc* — Fraction of processor consumed
- *%ec* — Percentage of entitled capacity consumed
- *lcs* — Total number of logical context switches

The **mpstat** command is a very useful command because it reports collection information for each logical CPU on your partition in a format that is clearly illustrated. You can even view SMT utilization by specifying the **-s** flag:

```
# mpstat -s 1

System configuration: lcpu=4 ent=0.4 mode=Uncapped

     Proc0           Proc1
      1.01%           0.02%
 cpu0    cpu1    cpu2    cpu3
  0.85%   0.16%   0.01%   0.01%
-------------------------------------------------------------------
     Proc0           Proc1
      0.74%           0.02%
 cpu0    cpu1    cpu2    cpu3
  0.56%   0.18%   0.01%   0.01%
```

topas (AIX-specific)

IBM has improved the **topas** command (part of the **bos.perf.tools** fileset) substantially in AIX 5.3. Before these changes, **topas** did not have the ability to capture historical data, nor was it enhanced for use in shared partitioned environments. (The command's **-L** flag now reports partitioned information.) By incorporating these changes to let you collect performance data from multiple partitions, IBM has really simplified the capability of **topas** as a performance management and capacity planning tool. The command's look and feel is quite similar to **top** and **monitor** (used in other Unix variants).

The **topas** utility displays all kinds of information on your screen in a text-based, graphical type of format. In its default mode, it provides a myriad of CPU, memory, and I/O information. Some recent changes:

- As of TL_4 of AIX 5.3, **topas** uses a daemon named **xmwlm**, which is automatically started from the **inittab**.
- As of TL_5 of AIX 5.3, the system keeps seven days of data as a default and records almost all the **topas** data that is displayed interactively, except for process and Workload Manager (WLM) information. You can use the **topasout** command to generate text-based reports. By specifying the **-C** flag, you can actually view monitoring information across all partitions in an IBM POWER system.

nmon

My favorite of all performance monitoring tools is **nmon**, which until recently was not an "officially" supported IBM tool; if you were going to send data to IBM for analysis, this was not the tool you would use. **nmon** is almost the perfect AIX analysis tool (it's also available now for Linux on POWER). The data it collects is available either from your screen or through reports, which you can run from **cron**. In the words of **nmon**'s creator, Nigel Griffiths, "Why use five or six tools when one free tool can give you everything you need?"

What attracts most people to **nmon** is that not only does it have a very efficient front-end monitor, but it also provides the ability (unlike **topas**) to capture data to a text file for graphing reports because the output is in a .csv (spreadsheet) format. In fact, moments after running an **nmon** session, you can actually view the nicely rendered charts in a Microsoft Excel spreadsheet, which you can hand off to senior management or other technical teams for further analysis. Further, in contrast to **topas**, I've never seen any performance-type overhead with this utility.

Using nmon for Historical Analysis

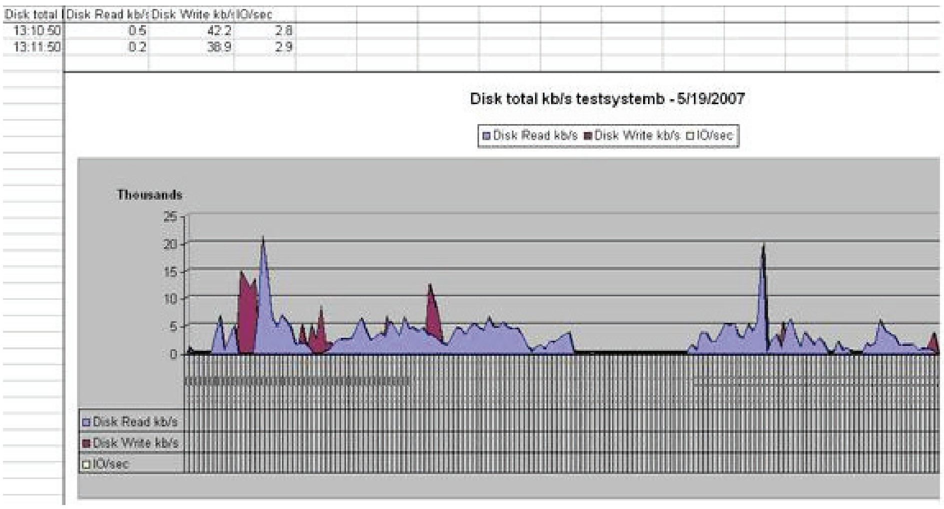
First, we'll tell **nmon** to create a file, name the run, and do data collection every 30 seconds for one hour (120 intervals):

```
# ./nmon -f -t -r test3 -s 30 -c 120
AIX version 5.3.0.0 and starting up nmon nmon_aix5
```

When monitoring is completed, we'll sort the file:

```
# sort -A p682e_pub_071224_1411.nmon > lpar30p682e_pub_071224_411.csv
```

Now, we can FTP the spreadsheet to a PC and open it up. Start the nmon analyzer, and click on **Analyze nmon data**. Enter the location of the file, wait about 20 seconds, and you'll see your **nmon** data in all its glory! Figure 5.1 shows some sample output from the nmon analyzer.



Disk total	Disk Read kb/s	Disk Write kb/s	IO/sec
13:10:50	0.5	42.2	2.8
13:11:50	0.2	38.9	2.9

Figure 5.1: Sample nmon analyzer output

The **nmon** analyzer is an awesome tool, written by Stephen Atkins, that graphically presents data (CPU, memory, network, or I/O) from an Excel spreadsheet. Perhaps the only drawback that prevents it from being perceived as an enterprise type of tool is that it lacks the ability to gather statistics about large numbers of LPARs at once (although it now has a partition-viewing capability similar to that of **topas**). The analyzer is not a database, nor was it meant to be. That is where a tool such as Ganglia helps; this utility has actually received the blessing of Nigel Griffiths as the tool that can integrate **nmon** analysis.

You can download the nmon analyzer for free from *http://www.ibm.com/developerworks/aix/library/au-nmon_analyser*. For more information about Ganglia, see *http://ganglia.info*.

ps (Unix-generic)

```
ps [-ANPaedfklmMZ] [-n namelist] [-F Format] [-o
   specifier[=header],...] [-p proclist][-G|-g grouplist] [-t
   termlist] [-U|-u userlist] [-c classlist] [ -T pid] [ -L pidlist]

ps [aceglnsuvwxU] [t tty] [processnumber]
```

The **ps** command shows the current status of processes. Upon viewing the syntaxes shown above, the first question you may have is, why the two sets of usage parameters? To make a long story short, the answer has to do with the basic history of Unix — the old Berkeley versus System V (now referred to as X/Open Standards) wars. As we discussed in Chapter 2, AIX is a hybrid of sorts, and it contains both flavors of Unix. Most of you are probably more familiar with the X/Open Standards usage of **ps** (e.g., **ps -ef**), which is the first usage shown above.

How can you best use **ps** in CPU systems monitoring? In other words, how can you identify processes that are taking an inordinate amount of CPU time? If you can find these processes, you can take action on them. I like using the Berkeley syntax better here; the information it provides is in a nicer, more presentable format. Let's look at **ps ux**, which displays the CPU execution time of processes:

```
# ps ux | more

USER        PID %CPU %MEM   SZ  RSS   TTY STAT     STIME  TIME COMMAND
root       8196  0.1  0.0  384  384    -    A  08:45:25  1:02 wait
root      53274  0.0  0.0  384  384    -    A  08:45:25  0:30 wait
root      86118  0.0  0.0  504  512    -    A  08:45:27  0:08 /usr/sbin/syncd
root     299158  0.0  0.0  472  500    -    A  08:45:44  0:06 /usr/sbin/getty
root      69666  0.0  0.0  960  960    -    A  08:45:25  0:04 gi
root          0  0.0  0.0  384  384    -    A  08:45:25  0:04 swappe
root      57372  0.0  0.0  384  384    -    A  08:45:25  0:02 wait
root      61470  0.0  0.0  384  384    -    A  08:45:25  0:02 wait
root     286880  0.0  0.0  900  928    -    A  08:45:44  0:01 /usr/bin/xmwlm-
root     258190  0.0  0.0 1216 1216    -    A  08:45:35  0:01 rpc.lock
root     151642  0.0  0.0  512  512    -    A  08:45:27  0:01 rtcmd
root     233606  0.0  0.0  840  956    -    A  08:45:44  0:00 /usr/sbin/sshd
```

This **ps** command uses two key parameters:

- **u** — Displays user-oriented output about each process: the *USER* (user), *PID* (process ID), *%CPU* (CPU time used), *%MEM* (memory used), *SZ* (size of process core image), *RSS* (resident set size), *TTY* (controlling terminal name), *STAT* (process state), *STIME* (start time), *TIME* (total run time), and *COMMAND* (executed command) fields.
- **x** — Displays processes without a controlling terminal in addition to processes with a controlling terminal. To see processes that don't include daemons, substitute **a** for **x**.

For our purposes, the most important field of the **ps** output is *%CPU*. This field reports the percentage of CPU time that the process has used since it started.

Tracing Tools

Tracing tools come in handy when you want to drill down further to analyze processes that are causing bottlenecks. Among these tools are **curt**, **splat**, **tprof**, **trace**, and **trcrpt**. We'll use the **tprof** and **trace** tools here.

tprof

```
tprof [ -c ] [ -C { all | cpuidslist } ] [ -d ] [ -D ] [ -e ]

  { [ -E { ALIGNMENT | EMULATION | ISLBMISS | DSLBMISS | PM_<event> } ]
  [ -f interval ] } [ -F ] [ -j ] [ -J profilehook ] [ -k ] [ -l ]
  [ -L objectslist ] [ -m objectslist ] [ -M sourcepathlist ]
  [ -p processlist ] [ -P { all | pidslist } ] [ -s ]

  [ -S searchpathlist ] [ -t ] [ -T buffersize ] [ -u ] [ -v ]
  [ -V verbosefilename ] [ -I ] [ -N ] { [-z] [-Z] | -R }
  { { -r rootstring } [ -X { xmloptions } ] |
    { { [ -A { all | cpuidslist } ] [-n] } [ -r rootstring ] -x command }
  }
```

The **tprof** command reports CPU usage for both individual programs and the system as a whole. The output provides an estimate of the amount of CPU time spent for each process that was executing while **tprof** was running. It also contains an estimate of the amount of CPU time spent in each of the kernel address spaces: the kernel address space, the user address space, and shared library address spaces.

You can use **tprof** to view a basic global program and thread-level summary by running the command in the following fashion:

```
# tprof -x sleep 20
Mon Dec 24 18:55:54 2

System: AIX 5.3 Node: lpar30p682e_pub Machine: 00CED82E4C0
Starting Command sleep 2
stopping trace collection.
Generating sleep.prof
root@lpar30p682e_pub[/]
```

Let's view the file (**sleep.prof**) that we just created:

```
# more sleep.prof
Configuration information


=========================
System: AIX 5.3 Node: lpar30p682e_pub Machine: 00CED82E4C00
```

Next, let's use the **trace** command to run a manual trace:

```
    /usr/bin/trace -ad -M -L 109113753 -T 500000 -j
   000,00A,001,002,003,38F,005,006,134,139,5A2,5A5,465,234, -o -
Total Samples = 1088
Traced Time = 20.02s (out of a total execution time of 20.02s)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Process                              Freq  Total Kernel   User Shared  Other
=======                              ====  ===== ======   ==== ======  =====
wait                                    4  99.82  99.82   0.00   0.00   0.00
swapper                                 1   0.09   0.09   0.00   0.00   0.00
/usr/bin/tprof                          1   0.09   0.00   0.00   0.09   0.00
Total                                   6 100.00  99.91   0.00   0.09   0.00
Process                        PID       TID  Total Kernel   User Shared  Other
=======                        ===       ===  ===== ======   ==== ======  =====
wait                          8196      8197  44.58  44.58   0.00   0.00   0.00
swapper                          0         3   0.09   0.09   0.00   0.00   0.00
/usr/bin/tprof              418000    688307   0.09   0.00   0.00   0.09   0.00

=======                        ===       ===  ===== ======   ==== ======  =====
Total                                        100.00  99.91   0.00   0.09   0.00
```

The **tprof** command is an excellent tool for identifying runaway processes because these processes appear at the top of the output list.

Timing Tools

Two tools, **time** and **timex**, provide access to information about command execution time.

time

```
time [ -p ] Command [ Argument ... ]
```

The **time** command returns the total execution time of your program, including real time, user time, and system time. This information can be useful when you're trying to figure out the amount of time it takes for commands to execute. **time** works by counting the CPU ticks from the time the command was first started until the time it ends:

```
# time find ./ -depth 1>/dev/null

real    0m23.30s
user    0m0.22s
sys     0m2.10s
```

timex

```
timex [ -s ][ -o ][ -p [ -fhkmrt ] ] cmd
```

Without any flags, the **timex** command provides the same type of information as **time**, but with a prettier view. Used with the **-s** flag, it summarizes all system activity while the command is being executed. This spares you the task of starting up a **sar** or **vmstat** process while running a timing. For this reason alone, I like to use **timex**, and I've found it a very useful tool through the years.

```
# timex -s find ./ -depth 1>/dev/null

real 21.69
user 0.20

sys  2

AIX lpar30p682e_pub 3 5 00CED82E4C00    12/26/07
System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08    %usr    %sys    %wio   %idle   physc   %entc
08:40:30       5      33       0      62    0.17    43.2

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
08:40:30       0       0       0       0       0       0       0       0

System configuration: lcpu=4 mem=3072MB ent=0.40 mode=Uncapped
08:40:08   slots cycle/s fault/s  odio/s
08:40:30  392358    0.00   18.11    0.00

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
08:40:30       0       0       0       0       0       0

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
08:40:30   19659       8    5522    0.14    0.18   12407  308149

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08 cswch/s
08:40:30    5617

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08  iget/s lookuppn/s dirblk/s
08:40:30       0       8513        0
```

```
System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08 runq-sz %runocc swpq-sz %swpocc
08:40:30     1.3      95

System configuration:  mode=Uncapped
08:40:08  proc-sz     inod-sz     file-sz     thrd-sz
08:40:30  68/262144   0/170       387/1124    219/524288

System configuration: lcpu=4 ent=0.40 mode=Uncapped
08:40:08   msg/s  sema/s
08:40:30    0.00    0.00
```