Chapter 22

Performance & Tuning

HP-UX Handbook
Revision 13.00

FEEDBACK or QUESTIONS: please email essam.ackleh@hp.com
(please use subject syntax: *HP-UX Handbook v13.00 Chapter <YY> - <Feedback Title>*

TABLE OF CONTENTS

HP-UX Performance and Tuning

Serious performance issues are complex ,often involving the Operating System, and the physical resources which may not be HP components. In depth performance analysis is considered to be a Consulting engagement best addressed by the APS Customer Performance Team (CPT) which provides Performance Solutions and Consulting for complex computing environments using HP-UX and Linux.
Examples of Consulting engagements:

- Customer complains of application performance, no triage has been done
- Oracle is performing slowly
- Oracle performance issues where Oracle AWR/statspack needs to be analyzed
- Comparison with other vendors (worked better on IBM, SUN, PA-RISC)
- High User CPU by application
- High I/O service times (can be due to poorly configured SAN, or FS tuning)
- Customer running their own benchmark (iozone, dd, cp, ftp, etc)
- Customer migrates to new HP-UX system and performance expectation set by sales team is missed (for example, expectations were for 40% improvement, but customer only got a 20% improvement, but be careful, sometimes it's a product limitation, such as HPVM)

Please refer such cases to them at the following link :
http://intranet.hp.com/tsg/WW2/CPT/Pages/CustomerPerformanceTeam.aspx

Contractually we are obligated to deliver due diligence to identify :

- High System CPU
- Mini-hangs
- Performance issues with HP products (VxFS, LVM, NFS, UFC, HP-VM, , fbackup, etc)
- Performance problems after installing patches (sometimes misleading as the problem was caused by the reboot, not the patch install)
- High I/O service times (can be due to poor tests, FS tuning, VxFS bug, lock contention, etc)
- System out of memory (check for memory leaks, etc)
- Customer running their own benchmark and uncovers an HP-UX bug

To accurately identify these areas , standard performance tools should be used . If an OS defect is suspected, L2 may implement additional instrumentation sourced from the Worldwide Technical Expert Center and engage an L3 resource to facilitate resolution.
Among the standard tools are the System Activity Reporter (sar) , iostat , vmstat, ps , and top.
Some clients have OpenView products such as Glance , Measureware and PerfView. These products and their use are supported through the OpenView support Team.
First pass load and configuration should be collected to clarify the systems resources and

configuration under problem conditions. Samples should be of sufficient length and granularity to allow a statistically significant sample. As problems root cause may not be clear, it is prudent to capture a wide scope of data to analyze for potential resource contention .

Memory issues :

```
swapinfo -tam
vmstat -n 60 20
ps -elf |sort -rnk 10
sar -b 5 120
```

CPU

```
sar -Muq  5  120
ps -elf |sort -rnk 8
```

System Tables

```
sar -v 10  60
```

OS configuration :

```
kctune -v  ( if 11.11 kmtune)
more /etc/fstab
bdf -i
```

Memory

HP-UX utilizes both physical memory, RAM and disk memory, referred to as swap. While there are a number of recommendations for memory sizing , the following is a practical guideline:

- There should sufficient RAM to run all concurrent process and threads in RAM
- There should be sufficient device swap to capture a minimum crash dump (25% of RAM)
- There should be a sufficient quantity of device swap to cover all initial fork calls , i.e. process reserve memory .

The Global System Memory Map is comprised of the systems physical memory and configured swap.

The total address space for 32-bit architecture is 4 GB. For 64- bit, the kernel has access to 16 Terabytes out of 4 Exabytes. Although the 64-bit kernel in wide mode has 64-bit addressing capability, currently only 42 out of the possible 64 bits are used. This leaves gaps represented in the graphs below as shaded regions.

This space is broken down into four equal size sections called ***quadrants***. No memory segment may cross a quadrant boundary. For 32 bit no single process can access larger than 1GB of address space. For 64 bit , while the address space is considerably larger, however no process can access larger than 1 quadrant of address space.

How Memory is Mapped in HP –UX

Kernel Address Space Layout
In kernel mode the process context space registers (sr4-sr7) need to be set to kernel context. This allows the kernel to access its virtual address space using short pointers.

Quadrant 1
Unlike user processes, all the kernel **text** and **data** reside in Quadrant 1. Quadrant 2 does not contain a private area used for data. The kernel is privileged and its text and data are sharable amongst the various subsystems, as a result there is no need for specific space for private and shared data.

The 32- bit the virtual addresses in Quadrant 1 are managed by the **sysmap_32bit** resource map, for 64-bit **sysmap_64bit** manages the resource map. The kernel text and data reside in space id **KERNELSPACE**, this is currently defined as 0. The zero value was selected to allow equivalently mapped addresses .
Space register 4 (**sr4**) will be set to 0 when running in kernel mode.

Quadrant 2
The majority of Quadrant 2 is unused for kernel address space. The kernel sets is "process" context space register to 5 (**sr5**) to point to the **Uarea** space id of the currently running thread. This allows the kernel to access the process context and threads using short pointers.

As of HP-UX 11i, the Uarea takes up 16 memory pages of virtual address space. For 32-bit kernels the Uarea starts at address 0x7FFF0000, for 64-bit the Uarea starts at 0x400003FF FFFF0000. While 16 pages have been reserved for virtual addresses, currently only 4 pages are allocated for 32 –bit and 8 pages for 64-bit Uarea.

Quadrant 3
In Quadrant 3, the kernels space register is set to 6 (**sr6)** and points to the buffer cache, i.e. **sr6=bufcache_spacid**. All of the virtual address assignments for the buffer cache/file cache are in Quadrant 3. The kernel utilizes a *bitmap* called **bufmap** for 64-bit and **bufmap2** for 32-bit to manage the virtual addresses in Quadrant 3. Remaining memory can be allocated to shared object memory calls.

Quadrant 4
In kernel mode Quadrant 4 virtual addresses are primarily used for kernel I/O. For 32-bit kernel I/O addresses and user I/O addresses are the same. They both reside in space id 0. 64-bit kernel running, kernel I/O addresses and user addresses are not the same. Kernel I/O space id is 0 , and user I/O uses a different space id , if both kernel and user I/O needs to reference the same page , I/O aliasing must be done.

NOTE:
Memory is allocated on sequential basis; it must be allocated in contiguous segments. Due to this protocol, while there may be enough total memory free for a process , if there isn't a large enough contiguous segment , the call to memory will fail.
Whether in kernel or user mode, the system will allocate the first free segment that satisfies the request.

The following parameters define the maximum *individual segment* size for the resource:
All parameters are limited to quadrant boundaries for their bitness.
The amount of free contiguous space in a quadrant determines if the request for these resources can be satisfied.
maxdsiz -the maximum allowed size for the Data segment of a 32-bit process

maxdsiz_64bit -the maximum allowed size for the Data segment of a 64-bit process

maxssiz- the maximum allowed size for the kernel Stack segment of a 32-bit process

maxssiz_64 bit- the maximum allowed size for the kernel Stack segment of a 64-bit process

maxtsiz- the maximum allowed size for the Text segment of a 32-bit process

maxtsiz_64bit- the maximum allowed size for the Text segment of a 64-bit process

shmmax — maximum size (in bytes) for a System V shared memory segment

Text segments get first priority allocation in quadrant 1. Kernel Stack segments get first priority allocation in quadrant 2

Buffer Cache/File Cache
Recent improvements to the buffer/file cache performance allow larger caches to be used without the severe performance penalties seen in the past. In 11.31 the Dynamic Buffer Cache was replaced by the VxFS file cache. This operates similarly and has the same 50% default value. The parameters filecache_min and filecache_max define the range of this cache.
As with the buffer cache in earlier revisions, % rcache and %wcache should be greater than 90% in a well tuned system . The cache parameters are dynamic and require no reboot in 11.31

In modern systems , there are a number of factors to determine in sizing the caches.

- Is the system using a disk array / SAN that has onboard memory for caching read/ writes?
- Does the system use a database that contains an internal buffer pool ? (Oracle as an example)
- Does the system use Online JFS direct mounts , bypassing the cache?
- Are the I/O's larger than the cache threshold (discovered_direct_iosz) ?

Keep in mind that many modern disk arrays buffer their writes with onboard memory, also many databases use lockable memory to buffer within the System Global Area or SGA. Some databases do not use an SGA and may benefit from a large cache.

Note:
Buffers remain in the cache even after a file is closed , as they could be used again in the future.

Some database vendors recommend locking the cache down , i.e. making the minimum and maximum cache size the same, this would be referred to as a static cache. Trade-offs are associated with either a static or dynamic buffer cache. If memory pressure exists, a static cache cannot be reduced, potentially causing more important pages to be swapped out or processes deactivated. A dynamic cache expands very rapidly, but contracts very slowly and only when memory pressure exists. It is possible to bypass either static or dynamic caches , in some instances this allows for faster disk I/O .

This can be accomplished with the Online JFS mount options:
mincache=direct convosync=direct
These should be used together to avoid mixed I/O performance issues.
Example:
mount -F vxfs -o remount, mincache=direct, convosync=direct /dev/vgtest/lvol1 /test

to return to default :

mount -F vxfs -o remount /dev/vgtest /lvol1 /test

If the I/O size is greater than 256KB the system will bypass the cache and use discovered_direct_io. This cache threshold can be tuned using vxtunefs , example:
vxtunefs -o discovered_direct_iosz=10485760 /u08

Raw logical volumes bypass the cache, while this has been historically faster in the past, improvements in I/O handling no longer account for a substantial difference in well tuned systems.

To monitor the use of the cache run the following command :
```
sar –b 5 100
```
You will see output similar to :
```
bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
  0            95          100        1        2                54        0
0
```
In this example we can see that while the system is reading everything it has written to in cache, it is only writing 54%. If this is consistent behavior the size of the cache can be reduced by at 45% of the current value.

Large and volatile caches can still have a negative impact on performance. It is prudent to have the latest cache performance patch configured on systems with large caches.

Insufficient memory resources are a major cause of performance problems, and should be the first area to check. To determine virtual memory configuration run the following command:

Tools for monitoring memory allocation

To determine the amount and usage of swap, the **swapinfo** command can be executed:

```
# swapinfo –tam
Mb Mb Mb PCT START/ Mb TYPE
AVAIL USED FREE USED LIMIT RESERVE PRI NAME
 dev 8192 0 8192 0% 0 - 1 /dev/vg00/lvol2
dev 8192 0 8192 0% 0 - 1 /dev/vg00/lvol9
dev 34520 0 34520 0% 0 - 1 /dev/vg13/lvol1
reserve - 31358 -31358
 memory 49020 21337 27683 44%
total 99924 52695 47229 53% - 0 –
```

The fields represent:
dev -Paging space residing on a mass storage device. The *Used* amount reflects only what is being paged to disk .

reserve – Process paging space on reserve. Upon *fork* the *estimated* size in memory for a process is locked in this pool until process termination This is only true of device swap .

memory -also known as *pseudo-swap* . This area of RAM is used for direct memory access for on processor memory calls and when device swap is exhausted to serve as process reserve memory.

At system startup ,*swapspc_cnt* and *swapmem_cnt* are initialized to the total amount of swap space and pseudo-swap available, i.e. *swapspc_max* and *swapmem_max* respectively.
Any time the *swapon* call is made *swapspc_cnt is* adjusted by the amount of swap added.
A process reserves swap simply by decrementing these two counters.

The kernel doesn't actually use disk blocks until they are needed. Device swap is used first, until **swapspc_cnt=0 .** When a process actually has to allocate swap the system knows it has enough room to do so as it has been allocated at the time of the fork call.

If there is no device or file system swap available , the system allocates pseudo-swap as a last resort. It decrements *swapspc_cnt* and keeps track of regions using pseudo-swap via a *pswaplist*. Pseudo-swap is either free or pseudo-allocated , ***it is never reserved.***

If there is an insufficient amount of available memory to fork you will receive an error :
cannot fork, not enough virtual memory.

If this error is received , you will need to allocate more device swap. This should be configured on a disk with no other swap partitions, and ideally of the same size and priority of existing swap logical volumes to enable interleaving.

In HP-UX , the only reason a process will die for lack of swap space is because of data area growth (using *sbrk*),stack growth, or if lazy swap is used.

If pseudo-swap is disabled by setting *swapmem_on* to 0, there will typically be a need to increase the amount of device swap in the system to accommodate paging and reserve area. Ideally, in a modern system paging to disk should be avoided. If there is significant paging to disk and the buffer cache/file cache has been adjusted to avoid contention , adding RAM would be advisable for maximum performance.

To monitor memory over time , the vmstat utility is very useful. It can be used to track memory use over time. The count and iteration fields are similar to sar . By focusing on the avm and free fields , you can determine if the amount of active open process memory use , and the memory on the free list are a concern over time. Paging activity can also be assessed.

Example : vmstat –n 60 20 - produces a sample once a minute for 20 minutes.

```
VM
       memory                  page faults
avm     free     re   at  pi  po  fr  de  sr  in    sy    cs
4863083 3869884  1388 9   0   0   0   0   0   7867 91843 4134
```

It will also provide CPU use at the interval:

```
Cpu       procs
us sy id   r b w
12 7 82   2 1 0
 24 5 71
 16 6 78
 12 6 81
34 5 60
 18 8 74
 19 7 74
 23 6 70
```

The column headings and the meaning of each column are:

procs

Information about numbers of processes in various states.

r -In run queue
b -Blocked for resources (I/O, paging, etc.)
w -Runnable or short sleeper (< 20 secs) but swapped

memory

Information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes that are running or have run in the last 20 seconds.

avm -Active virtual pages
free -Size of the free list

page (4096 bytes)

Information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

re -Page reclaims (without -S)
at -Address translation faults (without -S)
si -Processes swapped in (with -S)
so -Processes swapped out (with -S)
pi -Pages paged in
po -Pages paged out
fr -Pages freed per second
de -Anticipated short term memory shortfall
sr -Pages scanned by clock algorithm, per second

faults

Trap/interrupt rate averages per second over last 5 seconds.

in -Device interrupts per second (nonclock)
sy -System calls per second
cs -CPU context switch rate (switches/sec)

cpu

Breakdown of percentage usage of CPU time for the active processors

us -User time for normal and low priority processes
sy -System call time
id -CPU idle

Disk I/O

Disk I/O efficiency is effected by a number of factors:

- **The amount of reads and writes pending in the run queue**
- **The cumulative load on the controller, and it's maximum capacity**
- **The amount of buffer/file cache available**
- **The size of I/O's**
- **Mount options**
- **Patch level**
- **Memory contention**
- **CPU resources**

To determine Disk I/O load , the system activity reporter is a good starting point, to obtain a statistically significant sample it is prudent to gather data over sufficient time:
Example :

```
sar –d  5 100
```

This will provide sufficient granularity and averages to determine initial issues
The final output will look similar to :

```
               device  %busy  avque  r+w/s   blks/s  avwait  avserv
 Average    c2t6d0    1.50    0.51      4      66    0.06    6.31
Average    c7t0d2    2.62   19.58    122   15023    3.44    1.27
```

These fields represent the key load metrics :
%busy Portion of time device was busy servicing a request

avque Average number of requests outstanding for the device
The normal value for this is 0.50 or half of the SCSI queue depth.

r+w/s Number of data transfers per second (read and writes) from and to the device

blks/s Number of bytes transferred (in 512-byte units) from and to the device
When this exceeds 512 , I/O by file system should be assessed in Glance to determine if the cache threshold (256KB) is being exceeded. If so adjust *discovered _direct_iosz* appropriately.
avwait Average time (in milliseconds) that transfer requests waited idly on queue for the device

avserv Average time (in milliseconds) to service each transfer request (includes seek, rotational latency, and data transfer times) for the device.

When average wait (avwait) is greater than average service time (avserv) it indicates the disk can't keep up with the load during that sample.

When the average queue length exceeds the norm of 0.50 it is an indication of jobs stacking up. When it reaches 1.0 the run queue is at saturation.

If the avserv is under 15mS ,the SCSI run queue depth can be increase from the default of 8. In 11.23 and prior releases this is a global kernel setting : **scsi_max_qdepth** .

In 11.31 it can be set via the **scsimgr** utility , specifying the **max_q_depth** attribute . Once the service times exceed 15mS and the run queue is greater than 1.0 it is best to address the Physical layer overload , i.e. spread the load across more physical paths.

If disks are observed with no apparent load but high service times , check the other disks associated with that controller to see if they are experiencing high run queues. This can be a case of controller saturation.

These conditions are considered to be a bottleneck. It is prudent to keep in mind how long these conditions last. If the queue flushes, or the avwait clears in a reasonable time, (i.e. 5 seconds), it is not a cause for concern.

To determine the amount of reads and writes sar –c can be used :

```
# sar – c  5 100
              scall/s  sread/s  swrit/s   fork/s   exec/s  rchar/s  wchar/s
Average    5449      521      290     0.84     0.66  1428448  3682636
```

Vxtunefs has options to address specific read and write parameter values.
Refer to the man (1M) **vxtunefs** for details
Keep in mind that the more jobs in a queue, the greater the effect on wait on I/O even if they are small. Large jobs, those greater than 1000 blks/s will also affect throughput.

Also consider the type of disks being used. Modern disk arrays are capable of handling very large amounts of data in very short processing times. Processing loads of 5000 blks/s or greater in under 10mS. Older standard disks may show far less capability.

If a bottleneck is identified, run:

```
# strings /etc/lvmtab
```

to identify the volume group associated with the disks.

# cat /etc/fstab
to determine the file system type associated with the lvol/ mountpoint
# ***bdf***
to see if this volumes file systems are reaching capacity full (> 85%)

If installed Glance can be used to identify I/O by file system

How to improve disk I/O ?

1. Reduce the volume of data on the disk path to till the run queue is less than 1.0
2. Stripe the data across disks to improve I/O speed and decrease path overload.
3. If you are using Online JFS , run fsadm –e to defragment the extents.
4. Tune the buffer cache / file cache:

   Check the cache use with sar – b , ideally the writes to cache **(%wcache)** should be at least 75%.
   If the %wcache is low , consider the size of the I/O's and the cache threshold before tuning the cache size. If these are OK , then the cache size can be reduced .

   Example:
   ```
   05:00:00 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
   Average      0      195      100    131      216    39       0        0
   ```

   Most data bases buffer internally , for example Oracle uses the SGA buffer pool . Modern arrays also have onboard cache , these two areas should be taken into consideration when sizing the dynamic buffer cache/file cache

5. Consider direct mount options, these are available only with the Online JFS product.
   The VxFS direct mount options are mincache=direct and convosync=direct, these should be used together to avoid mixed I/O This can be tested without unmounting the file system :

   To change mount options on the fly:
   mount -F vxfs -o remount, mincache=direct, convosync=direct /dev/vg_name/lvol_name /mount_point
   to return to default : ***mount -F vxfs -o remount /dev/vg_name/lvol_name /mount_point***

6. If you are using raw logical volumes, consider implementing asynchronous I/O.

   The difference between the async I/O and synchronous I/O is that async does not wait for confirmation of the write before moving on to the next task. This does increase the speed of the disk performance at the expense of robustness. Synchronous I/O waits for acknowledgement of the write (or fail) before continuing on. The write can have physically taken place or could be in the buffer cache but in either case, acknowledgement has been sent. In the case of async, no waiting.

To implement asynchronous I/O on HP-UX for raw logical volumes:

* set the async_disk driver (Asynchronous Disk Pseudo Driver)to IN
in the HP-UX Kernel, this will require generating a new kernel and rebooting .
* create the device file:

# mknod /dev/async c 101 0x00000#

#=the minor number can be one of the following values:

```
0x000000 default-immediate reporting will not be used
a cache flush will be done before posting an IO operation complete
0x000001 enable immediate reporting
0x000002 flush the CPU cache after reads
0x000004 allow disks to timeout
0x000005 is a combination of 1 and 4
0x000007 is a combination of 1, 2 and 4
```

Note:

HP does not make minor number recommendations, have the client contact the database vendor or product vendor to determine the correct minor number for your application.

Change the ownership to the appropriate group and owner :

```
# chown oracle:dba /dev/async
```

change the permissions:

```
# chmod 660 /dev/async
# vi /etc/privgroup
```

add 1 line : dba MLOCK

give the group MLOCK privileges : /usr/sbin/setprivgrp MLOCK

To verify if a group has the MLOCK privilege execute:

```
# /usr/bin/getprivgrp
```

The default number of available ports for asynchronous disks is 50 , this is tuned with the kernel parameter ***max_async_ports***, if greater than 50 disks are being used, this parameter needs to be increased .

Check the system for the most current I/O and file system related patches.
Ideally patches should be no more than 1 year from current level.

Load Balancing

Load balancing policy depends on the environment, there are 4 choices :

- **round_robin: A**ppropriate when all the paths to the device have similar I/O turn-around characteristics.
- **least_cmd_load**: The LUN path with the least number of active I/O requests is selected to execute the next I/O. This policy is appropriate when the paths to the LUN exhibit asymmetric latency characteristics. The load is distributed to optimize the bandwidth on each LUN path.
- **cl_round_robin** (*cell aware round robin*): This load balancing policy is applicable to HP cell-based platforms. The LUN paths are selected in a round robin manner within the locality of CPU on which the I/O was initiated, to ensure that memory access latencies are optimized.

- **preferred_path:** The I/O path set in the preferred_path attribute is preferrably used for I/O transfer. If this I/O path is not available or if the preferred_path attribute was not set, any other path is selected for I/O transfer. This policy is useful for certain disk arrays, which may exhibit some performance degradation if I/Os are transferred via several I/O paths to a LUN simultaneously.

CPU

Once we have determined that the memory resources are adequate, we need to address the processors. We need to determine how many processors there are, what speed they run at and what load they are under during a variety of system loads .

To find out cpu load on a multi-processor system, run :
example: sar –Mu 5 100
This will produce 100 data points 5 seconds apart.
The output will look similar to :

```
11:20:05    cpu   %usr   %sys   %wio   %idle
11:20:10      0      1      1      0     99
              1     17     83      0      0
         system      9     42      0     49
```

After all samples are taken an average is printed
This will return data on the cpu load for each processor:
cpu - cpu number (only on a multi-processor system and used with the -M option)
%usr - user mode
%sys- system mode
%wio - idle with some process waiting for I/O
(only block I/O, raw I/O, or VM pageins/swapins indicated)
%idle - other idle

The %wio is a subset of idle time. The data for the metric is collected during a clock tick. The kernel checks each processor to determine whether it is running a thread or process or is idle. If it is idle, it checks to see if there is any outstanding IO for that processor. If there is, the time gets counted as %wio, if not the time is counted as idle time.

If the cpu is busy, the time is attributed either to user cpu or system cpu based on the process. There is no check on I/O at all. A busy system doing lots of I/O can often show less wio than an idle system doing less actual IO.

If the system is a single processor system under heavy load the CPU bottleneck may be unavoidable. The cpu run queue is a good indicator of actual CPU load , the %runocc provides an idea of how much bandwidth the CPU has , when it reaches 100 the CPU is at saturation.

To find out what the run queue load is, run :
Example:

```
sar –q 5 100
         runq-sz %runocc swpq-sz %swpocc
10:06:36     0.0      0    0.0      0
10:06:41     1.5     40    0.0      0
10:06:46     3.0     20    0.0      0
10:06:51     1.0     20    0.0      0
Average      1.8     16    0.0      0
```

runq-sz - Average length of the run queue(s) of processes (in memory and runnable)
%runocc - The percentage of time the run queue(s) were occupied by processes
(in memory and runnable) maximum is 100
swpq-sz - Average length of the swap queue of runnable processes
(processes swapped out but ready to run)

These cpu reports can be combined using sar -Muq .

Check the output of ps –elf for the cumulative CPU time of application and OS based processes , and their priorities. Some alteration of scheduling policies may be beneficial for application performance.

Tuning of the system caches can also optimize CPU efficiency. Default values for ***vx_ninode* , *vxfs_ifree_timelag* , *filecache_max*** can all lead to excessive CPU overhead .

Scheduling

There are instances when the default POSIX scheduler is inadequate for applications to run efficiently . HP-UX has a number of schedulers available, altering the default scheduler always come with the risk of making some processes or threads run slower due to priority based context switching.

:

POSIX real-time schedulers:	SCHED_FIFO
	SCHED_RR
	SCHED_RR2
HP-UX real-time scheduler:	SCHED_RTPRIO
HP-UX timeshare scheduler:	SCHED_HPUX
	SCHED_NOAGE

The default `priority` range of each scheduler is as follows:

scheduler	highest priority	lowest priority
SCHED_FIFO	31	0
SCHED_RR	31	0
SCHED_RR2	31	0
SCHED_RTPRIO	0	127
SCHED_NOAGE	178	255
SCHED_HPUX	N/A	N/A

The most commonly used alternate scheduling policies are SCHED_NOAGE , this allows processes to hold their priority while on CPU , avoiding priority based context switching. Typically this is set at priority 178, which at the top of the system call range.

Under normal conditions a process will age , i.e. it's priority will become less important through it's ***nice*** range while on CPU . As soon as it becomes less important in priority that any other process in the queue , the system forces a context switch. If there are many similar priority processes running in a given CPU run queue , this can cause excessive context switching and an insufficient amount of contiguous CPU time to get much done.

SCHED_RTPRIO is also somewhat popular for large database environments. This allows the specified processes to run under normal System Call range.

Note: Higher numerical values for the *priority* represent higher priorities under POSIX real-time schedulers, whereas lower numerical values for the *priority* represent higher priorities under HP-UX real-time and timeshare schedulers

In presence of processor sets , the application execution is restricted to processors in the application's processor set. The threads in different processor sets do not compete with one another for processors based on their scheduling policy and priority values. The scheduler looks only at threads assigned to a processor's processor set to choose the next thread to run.

RETURN VALUE

rtsched returns exit status:

0 if *command* is successfully scheduled or if *pid*'s real-time priority is successfully changed;

1 if *command* is not executable, *pid* does not exist, or *priority* is not within the priority range for the corresponding scheduler;

2 if *command* (*pid*) lacks real-time capability, or the invoker's effective user ID is not a user who has appropriate privileges, or the real or effective user or the real or effective user ID does not match the real or saved user ID of the process being changed; or

5 if rtsched encounters an internal error or is not supported by this release.
For File System tuning, please refer to Chapter 14 on JFS.