16.4. Managing Kernel Resources

A large PostgreSQL installation can quickly exhaust various operating system resource limits. (On some systems, the factory defaults are so low that you don't even need a really "large" installation.) If you have encountered this kind of problem, keep reading.

16.4.1. Shared Memory and Semaphores

Shared memory and semaphores are collectively referred to as "System V IPC" (together with message queues, which are not relevant for PostgreSQL). Almost all modern operating systems provide these features, but not all of them have them turned on or sufficiently sized by default, especially systems with BSD heritage. (For the Windows port, PostgreSQL provides its own replacement implementation of these facilities.)

The complete lack of these facilities is usually manifested by an Illegal system call error upon server start. In that case there's nothing left to do but to reconfigure your kernel. PostgreSQL won't work without them.

When PostgreSQL exceeds one of the various hard IPC limits, the server will refuse to start and should leave an instructive error message describing the problem encountered and what to do about it. (See also Section 16.3.1.) The relevant kernel parameters are named consistently across different systems; Table 16-1 gives an overview. The methods to set them, however, vary. Suggestions for some platforms are given below. Be warned that it is often necessary to reboot your machine, and possibly even recompile the kernel, to change these settings.

Table 16-1. System V IPC parameters

Name	Description	Reasonable values
SHMMAX	Maximum size of shared memory segment (bytes)	at least several megabytes (see text)
SHMMIN	Minimum size of shared memory segment (bytes)	1
SHMALL	Total amount of shared memory available (bytes or pages)	if bytes, same as SHMMAX; if pages, ceil(SHMMAX/PAGE_SIZE)
SHMSEG	Maximum number of shared memory segments per process	only 1 segment is needed, but the default is much higher
SHMMNI	Maximum number of shared memory segments system-wide	like SHMSEG plus room for other applications
SEMMNI	Maximum number of semaphore identifiers (i.e., sets)	at least ceil(max_connections / 16)
SEMMNS	Maximum number of semaphores system-wide	ceil(max_connections / 16) * 17 plus room for other applications
SEMMSL	Maximum number of semaphores per set	at least 17

Name	Description	Reasonable values
SEMMAP	Number of entries in semaphore map	see text
SEMVMX	Maximum value of semaphore	at least 1000 (The default is often 32767, don't change unless forced to)

The most important shared memory parameter is SHMMAX, the maximum size, in bytes, of a shared memory segment. If you get an error message from shmget like Invalid argument, it is likely that this limit has been exceeded. The size of the required shared memory segment varies depending on several PostgreSQL configuration parameters, as shown in Table 16-2. You can, as a temporary solution, lower some of those settings to avoid the failure. As a rough approximation, you can estimate the required segment size as 500 kB plus the variable amounts shown in the table. (Any error message you might get will include the exact size of the failed allocation request.) While it is possible to get PostgreSQL to run with SHMMAX as small as 1 MB, you need at least 4 MB for acceptable performance, and desirable settings are in the tens of megabytes.

Some systems also have a limit on the total amount of shared memory in the system (SHMALL). Make sure this is large enough for PostgreSQL plus any other applications that are using shared memory segments. (Caution: SHMALL is measured in pages rather than bytes on many systems.)

Less likely to cause problems is the minimum size for shared memory segments (SHMMIN), which should be at most approximately 500 kB for PostgreSQL (it is usually just 1). The maximum number of segments system-wide (SHMMNI) or per-process (SHMSEG) are unlikely to cause a problem unless your system has them set to zero.

PostgreSQL uses one semaphore per allowed connection (max_connections), in sets of 16. Each such set will also contain a 17th semaphore which contains a "magic number", to detect collision with semaphore sets used by other applications. The maximum number of semaphores in the system is set by SEMMNS, which consequently must be at least as high as max_connections plus one extra for each 16 allowed connections (see the formula in Table 16-1). The parameter SEMMNI determines the limit on the number of semaphore sets that can exist on the system at one time. Hence this parameter must be at least ceil(max_connections / 16). Lowering the number of allowed connections is a temporary workaround for failures, which are usually confusingly worded No space left on device, from the function semget.

In some cases it might also be necessary to increase SEMMAP to be at least on the order of SEMMNS. This parameter defines the size of the semaphore resource map, in which each contiguous block of available semaphores needs an entry. When a semaphore set is freed it is either added to an existing entry that is adjacent to the freed block or it is registered under a new map entry. If the map is full, the freed semaphores get lost (until reboot). Fragmentation of the semaphore space could over time lead to fewer available semaphores than there should be.

The SEMMSL parameter, which determines how many semaphores can be in a set, must be at least 17 for PostgreSQL.

Various other settings related to "semaphore undo", such as SEMMNU and SEMUME, are not of concern for PostgreSQL

Linux

The default settings are only suitable for small installations (the default max segment size is 32 MB). However the remaining defaults are quite generously sized, and usually do not require changes. The max segment size can be changed via the sysctl interface. For example, to allow 128 MB, and explicitly set the maximum total shared memory size to 2097152 pages (the default):

```
$ sysctl -w kernel.shmmax=134217728
$ sysctl -w kernel.shmall=2097152
```

In addition these settings can be saved between reboots in /etc/sysctl.conf.

Older distributions may not have the sysctl program, but equivalent changes can be made by manipulating the /proc file system:

```
$ echo 134217728 >/proc/sys/kernel/shmmax
$ echo 2097152 >/proc/sys/kernel/shmall
```

Table 16-2. Configuration parameters affecting PostgreSQL's shared memory usage

Name	Approximate multiplier (bytes per increment)
max_connections	400 + 270 * max_locks_per_transaction
max_prepared_transactions	600 + 270 * max_locks_per_transaction
shared_buffers	8300 (assuming 8K BLCKSZ)
wal_buffers	8200 (assuming 8K XLOG_BLCKSZ)
max_fsm_relations	70
max_fsm_pages	6

16.4.2. Resource Limits

Unix-like operating systems enforce various kinds of resource limits that might interfere with the operation of your PostgreSQL server. Of particular importance are limits on the number of processes per user, the number of open files per process, and the amount of memory available to each process. Each of these have a "hard" and a "soft" limit. The soft limit is what actually counts but it can be changed by the user up to the hard limit. The hard limit can only be changed by the root user. The system call setrlimit is responsible for setting these parameters. The shell's built-in command ulimit (Bourne shells) or limit (csh) is used to control the resource limits from the command line. On BSD-derived systems the file /etc/login.conf controls the various resource limits set during login. See the operating system documentation for details. The relevant parameters are maxproc, openfiles,

and `datasize`. For example:

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
...
```

(`-cur` is the soft limit. Append `-max` to set the hard limit.)

Kernels can also have system-wide limits on some resources.

- On Linux `/proc/sys/fs/file-max` determines the maximum number of open files that the kernel will support. It can be changed by writing a different number into the file or by adding an assignment in `/etc/sysctl.conf`. The maximum limit of files per process is fixed at the time the kernel is compiled; see `/usr/src/linux/Documentation/proc.txt` for more information.

The PostgreSQL server uses one process per connection so you should provide for at least as many processes as allowed connections, in addition to what you need for the rest of your system. This is usually not a problem but if you run several servers on one machine things might get tight.

The factory default limit on open files is often set to "socially friendly" values that allow many users to coexist on a machine without using an inappropriate fraction of the system resources. If you run many servers on a machine this is perhaps what you want, but on dedicated servers you may want to raise this limit.

On the other side of the coin, some systems allow individual processes to open large numbers of files; if more than a few processes do so then the system-wide limit can easily be exceeded. If you find this happening, and you do not want to alter the system-wide limit, you can set PostgreSQL's max_files_per_process configuration parameter to limit the consumption of open files.

16.4.3. Linux Memory Overcommit

In Linux 2.4 and later, the default virtual memory behavior is not optimal for PostgreSQL. Because of the way that the kernel implements memory overcommit, the kernel may terminate the PostgreSQL server (the master server process) if the memory demands of another process cause the system to run out of virtual memory.

If this happens, you will see a kernel message that looks like this (consult your system documentation and configuration on where to look for such a message):

```
Out of Memory: Killed process 12345 (postgres).
```

This indicates that the `postgres` process has been terminated due to memory pressure. Although existing database connections will continue to function normally, no new connections will be accepted. To recover, PostgreSQL will need to be restarted.

One way to avoid this problem is to run PostgreSQL on a machine where you

can be sure that other processes will not run the machine out of memory.

On Linux 2.6 and later, a better solution is to modify the kernel's behavior so that it will not "overcommit" memory. This is done by selecting strict overcommit mode via `sysctl`:

```
sysctl -w vm.overcommit_memory=2
```

or placing an equivalent entry in `/etc/sysctl.conf`. You may also wish to modify the related setting `vm.overcommit_ratio`. For details see the kernel documentation file `Documentation/vm/overcommit-accounting`.

Some vendors' Linux 2.4 kernels are reported to have early versions of the 2.6 overcommit `sysctl` parameter. However, setting `vm.overcommit_memory` to 2 on a kernel that does not have the relevant code will make things worse not better. It is recommended that you inspect the actual kernel source code (see the function `vm_enough_memory` in the file `mm/mmap.c`) to verify what is supported in your copy before you try this in a 2.4 installation. The presence of the `overcommit-accounting` documentation file should *not* be taken as evidence that the feature is there. If in any doubt, consult a kernel expert or your kernel vendor.