Red Hat Enterprise Linux Release 9.2 Manual Pages on 'sched_setaffinity.2' command

$ man sched_setaffinity.2

SCHED_SETAFFINITY(2) Linux Programmer's Manual SCHED_SETAFFINITY(2)

NAME

sched_setaffinity, sched_getaffinity - set and get a thread's CPU affinity mask

SYNOPSIS

#define _GNU_SOURCE /* See feature_test_macros(7) */

#include <sched.h>

int sched_setaffinity(pid_t pid, size_t cpusetsize,

const cpu_set_t *mask);

int sched_getaffinity(pid_t pid, size_t cpusetsize,

cpu_set_t *mask);

DESCRIPTION

A thread's CPU affinity mask determines the set of CPUs on which it is eligible to run. On a multiprocessor system, setting the CPU affinity mask can be used to obtain performance benefits. For example, by dedi? cating one CPU to a particular thread (i.e., setting the affinity mask of that thread to specify a single CPU, and setting the affinity mask of all other threads to exclude that CPU), it is possible to ensure maximum execution speed for that thread. Restricting a thread to run on a single CPU also avoids the performance cost caused by the cache invalidation that occurs when a thread ceases to execute on one CPU and then recommences execution on a different CPU.

A CPU affinity mask is represented by the cpu_set_t structure, a "CPU set", pointed to by mask. A set of macros for manipulating CPU sets is described in CPU_SET(3).

sched_setaffinity() sets the CPU affinity mask of the thread whose ID is pid to the value specified by mask. If pid is zero, then the call?ing thread is used. The argument cpusetsize is the length (in bytes) of the data pointed to by mask. Normally this argument would be speci?fied as sizeof(cpu_set_t).

If the thread specified by pid is not currently running on one of the CPUs specified in mask, then that thread is migrated to one of the CPUs specified in mask.

sched_getaffinity() writes the affinity mask of the thread whose ID is pid into the cpu_set_t structure pointed to by mask. The cpusetsize argument specifies the size (in bytes) of mask. If pid is zero, then the mask of the calling thread is returned.

RETURN VALUE

On success, sched_setaffinity() and sched_getaffinity() return 0 (but see "C library/kernel differences" below, which notes that the underly?ing sched_getaffinity() differs in its return value). On error, -1 is returned, and errno is set appropriately.

ERRORS

EFAULT A supplied memory address was invalid.

EINVAL The affinity bit mask mask contains no processors that are cur?rently physically on the system and permitted to the thread ac?cording to any restrictions that may be imposed by cpuset cgroups or the "cpuset" mechanism described in cpuset(7).

EINVAL (sched_getaffinity() and, in kernels before 2.6.9, sched_setaffinity()) cpusetsize is smaller than the size of the affinity mask used by the kernel.

EPERM (sched_setaffinity()) The calling thread does not have appropri?ate privileges. The caller needs an effective user ID equal to the real user ID or effective user ID of the thread identified by pid, or it must possess the CAP_SYS_NICE capability in the user namespace of the thread pid.

ESRCH The thread whose ID is pid could not be found.

VERSIONS

The CPU affinity system calls were introduced in Linux kernel 2.5.8. The system call wrappers were introduced in glibc 2.3. Initially, the glibc interfaces included a cpusetsize argument, typed as unsigned int. In glibc 2.3.3, the cpusetsize argument was removed, but was then re?stored in glibc 2.3.4, with type size_t.

CONFORMING TO

These system calls are Linux-specific.

NOTES

After a call to sched_setaffinity(), the set of CPUs on which the thread will actually run is the intersection of the set specified in the mask argument and the set of CPUs actually present on the system. The system may further restrict the set of CPUs on which the thread runs if the "cpuset" mechanism described in cpuset(7) is being used. These restrictions on the actual set of CPUs on which the thread will run are silently imposed by the kernel.

There are various ways of determining the number of CPUs available on the system, including: inspecting the contents of /proc/cpuinfo; using sysconf(3) to obtain the values of the _SC_NPROCESSORS_CONF and _SC_NPROCESSORS_ONLN parameters; and inspecting the list of CPU direc?tories under /sys/devices/system/cpu/.

sched(7) has a description of the Linux scheduling scheme.

The affinity mask is a per-thread attribute that can be adjusted inde?pendently for each of the threads in a thread group. The value re?turned from a call to gettid(2) can be passed in the argument pid. Specifying pid as 0 will set the attribute for the calling thread, and passing the value returned from a call to getpid(2) will set the attri?bute for the main thread of the thread group. (If you are using the POSIX threads API, then use pthread_setaffinity_np(3) instead of sched_setaffinity().)

The isolcpus boot option can be used to isolate one or more CPUs at boot time, so that no processes are scheduled onto those CPUs. Follow?

ing the use of this boot option, the only way to schedule processes onto the isolated CPUs is via sched_setaffinity() or the cpuset(7) mechanism. For further information, see the kernel source file Docu? mentation/admin-guide/kernel-parameters.txt. As noted in that file, isolcpus is the preferred mechanism of isolating CPUs (versus the al? ternative of manually setting the CPU affinity of all processes on the system).

A child created via fork(2) inherits its parent's CPU affinity mask.

The affinity mask is preserved across an execve(2).

C library/kernel differences

This manual page describes the glibc interface for the CPU affinity calls. The actual system call interface is slightly different, with the mask being typed as unsigned long *, reflecting the fact that the underlying implementation of CPU sets is a simple bit mask.

On success, the raw sched_getaffinity() system call returns the number of bytes placed copied into the mask buffer; this will be the minimum of cpusetsize and the size (in bytes) of the cpumask_t data type that is used internally by the kernel to represent the CPU set bit mask.

Handling systems with large CPU affinity masks

The underlying system calls (which represent CPU masks as bit masks of type unsigned long *) impose no restriction on the size of the CPU mask. However, the cpu_set_t data type used by glibc has a fixed size of 128 bytes, meaning that the maximum CPU number that can be repre? sented is 1023. If the kernel CPU affinity mask is larger than 1024, then calls of the form:

```
sched_getaffinity(pid, sizeof(cpu_set_t), &mask);
```

fail with the error EINVAL, the error produced by the underlying system call for the case where the mask size specified in cpusetsize is smaller than the size of the affinity mask used by the kernel. (De? pending on the system CPU topology, the kernel affinity mask can be substantially larger than the number of active CPUs in the system.)

When working on systems with large kernel CPU affinity masks, one must dynamically allocate the mask argument (see CPU_ALLOC(3)). Currently, the only way to do this is by probing for the size of the required mask using sched_getaffinity() calls with increasing mask sizes (until the call does not fail with the error EINVAL).

Be aware that CPU_ALLOC(3) may allocate a slightly larger CPU set than requested (because CPU sets are implemented as bit masks allocated in units of sizeof(long)). Consequently, sched_getaffinity() can set bits beyond the requested allocation size, because the kernel sees a few ad? ditional bits. Therefore, the caller should iterate over the bits in the returned set, counting those which are set, and stop upon reaching the value returned by CPU_COUNT(3) (rather than iterating over the num? ber of bits requested to be allocated).

EXAMPLES

The program below creates a child process. The parent and child then each assign themselves to a specified CPU and execute identical loops that consume some CPU time. Before terminating, the parent waits for the child to complete. The program takes three command-line arguments: the CPU number for the parent, the CPU number for the child, and the number of loop iterations that both processes should perform.

As the sample runs below demonstrate, the amount of real and CPU time consumed when running the program will depend on intra-core caching ef? fects and whether the processes are using the same CPU.

We first employ lscpu(1) to determine that this (x86) system has two cores, each with two CPUs:

```
$ lscpu | egrep -i 'core.*:|socket'
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
```

We then time the operation of the example program for three cases: both processes running on the same CPU; both processes running on different CPUs on the same core; and both processes running on different CPUs on different cores.

```
$ time -p ./a.out 0 0 100000000
real 14.75
```

```
    user 3.02
    sys 11.73
    $ time -p ./a.out 0 1 100000000
    real 11.52
    user 3.98
    sys 19.06
    $ time -p ./a.out 0 3 100000000
    real 7.89
    user 3.29
    sys 12.07
```

Program source

```
#define _GNU_SOURCE
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                        } while (0)
int
main(int argc, char *argv[])
{
    cpu_set_t set;
    int parentCPU, childCPU;
    int nloops;
    if (argc != 4) {
        fprintf(stderr, "Usage: %s parent-cpu child-cpu num-loops\n",
                argv[0]);
        exit(EXIT_FAILURE);
    }
    parentCPU = atoi(argv[1]);
    childCPU = atoi(argv[2]);
    nloops = atoi(argv[3]);
```

```
    CPU_ZERO(&set);

    switch (fork()) {
    case -1:            /* Error */
        errExit("fork");

    case 0:             /* Child */
        CPU_SET(childCPU, &set);
        if (sched_setaffinity(getpid(), sizeof(set), &set) == -1)
            errExit("sched_setaffinity");

        for (int j = 0; j < nloops; j++)
            getppid();

        exit(EXIT_SUCCESS);

    default:            /* Parent */
        CPU_SET(parentCPU, &set);
        if (sched_setaffinity(getpid(), sizeof(set), &set) == -1)
            errExit("sched_setaffinity");

        for (int j = 0; j < nloops; j++)
            getppid();

        wait(NULL);     /* Wait for child to terminate */
        exit(EXIT_SUCCESS);
    }
}
```

SEE ALSO

lscpu(1), nproc(1), taskset(1), clone(2), getcpu(2), getpriority(2), gettid(2), nice(2), sched_get_priority_max(2), sched_get_priority_min(2), sched_getscheduler(2), sched_setscheduler(2), setpriority(2), CPU_SET(3), get_nprocs(3), pthread_setaffinity_np(3), sched_getcpu(3), capabilities(7), cpuset(7), sched(7), numactl(8)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.