Rocky Enterprise Linux 9.2 Manual Pages on command 'CPU_ZERO_S.3'

$ man CPU_ZERO_S.3

CPU_SET(3) Linux Programmer's Manual CPU_SET(3)

NAME

CPU_SET, CPU_CLR, CPU_ISSET, CPU_ZERO, CPU_COUNT, CPU_AND, CPU_OR, CPU_XOR, CPU_EQUAL, CPU_ALLOC, CPU_ALLOC_SIZE, CPU_FREE, CPU_SET_S, CPU_CLR_S, CPU_ISSET_S, CPU_ZERO_S, CPU_COUNT_S, CPU_AND_S, CPU_OR_S, CPU_XOR_S, CPU_EQUAL_S - macros for manipulating CPU sets

SYNOPSIS

```
#define _GNU_SOURCE             /* See feature_test_macros(7) */
#include <sched.h>
void CPU_ZERO(cpu_set_t *set);
void CPU_SET(int cpu, cpu_set_t *set);
void CPU_CLR(int cpu, cpu_set_t *set);
int  CPU_ISSET(int cpu, cpu_set_t *set);
int  CPU_COUNT(cpu_set_t *set);
void CPU_AND(cpu_set_t *destset,
         cpu_set_t *srcset1, cpu_set_t *srcset2);
void CPU_OR(cpu_set_t *destset,
         cpu_set_t *srcset1, cpu_set_t *srcset2);
void CPU_XOR(cpu_set_t *destset,
         cpu_set_t *srcset1, cpu_set_t *srcset2);
int  CPU_EQUAL(cpu_set_t *set1, cpu_set_t *set2);
cpu_set_t *CPU_ALLOC(int num_cpus);
void CPU_FREE(cpu_set_t *set);
```

```
size_t CPU_ALLOC_SIZE(int num_cpus);

void CPU_ZERO_S(size_t setsize, cpu_set_t *set);

void CPU_SET_S(int cpu, size_t setsize, cpu_set_t *set);

void CPU_CLR_S(int cpu, size_t setsize, cpu_set_t *set);

int  CPU_ISSET_S(int cpu, size_t setsize, cpu_set_t *set);

int  CPU_COUNT_S(size_t setsize, cpu_set_t *set);

void CPU_AND_S(size_t setsize, cpu_set_t *destset,
        cpu_set_t *srcset1, cpu_set_t *srcset2);

void CPU_OR_S(size_t setsize, cpu_set_t *destset,
        cpu_set_t *srcset1, cpu_set_t *srcset2);

void CPU_XOR_S(size_t setsize, cpu_set_t *destset,
        cpu_set_t *srcset1, cpu_set_t *srcset2);

int  CPU_EQUAL_S(size_t setsize, cpu_set_t *set1, cpu_set_t *set2);
```

DESCRIPTION

The cpu_set_t data structure represents a set of CPUs. CPU sets are used by sched_setaffinity(2) and similar interfaces.

The cpu_set_t data type is implemented as a bit mask. However, the data structure should be treated as opaque: all manipulation of CPU sets should be done via the macros described in this page.

The following macros are provided to operate on the CPU set set:

CPU_ZERO()

Clears set, so that it contains no CPUs.

CPU_SET()

Add CPU cpu to set.

CPU_CLR()

Remove CPU cpu from set.

CPU_ISSET()

Test to see if CPU cpu is a member of set.

CPU_COUNT()

Return the number of CPUs in set.

Where a cpu argument is specified, it should not produce side effects, since the above macros may evaluate the argument more than once.

The first CPU on the system corresponds to a cpu value of 0, the next CPU corresponds to a cpu value of 1, and so on. No assumptions should be made about particular CPUs being available, or the set of CPUs being contiguous, since CPUs can be taken offline dynami? cally or be otherwise absent. The constant CPU_SETSIZE (currently 1024) specifies a value one greater than the maximum CPU number that can be stored in cpu_set_t.

The following macros perform logical operations on CPU sets:

CPU_AND()

Store the intersection of the sets srcset1 and srcset2 in destset (which may be one of the source sets).

CPU_OR()

Store the union of the sets srcset1 and srcset2 in destset (which may be one of the source sets).

CPU_XOR()

Store the XOR of the sets srcset1 and srcset2 in destset (which may be one of the source sets). The XOR means the set of CPUs that are in either srcset1 or srcset2, but not both.

CPU_EQUAL()

Test whether two CPU set contain exactly the same CPUs.

Dynamically sized CPU sets

Because some applications may require the ability to dynamically size CPU sets (e.g., to allocate sets larger than that defined by the standard cpu_set_t data type), glibc nowa? days provides a set of macros to support this.

The following macros are used to allocate and deallocate CPU sets:

CPU_ALLOC()

Allocate a CPU set large enough to hold CPUs in the range 0 to num_cpus-1.

CPU_ALLOC_SIZE()

Return the size in bytes of the CPU set that would be needed to hold CPUs in the range 0 to num_cpus-1. This macro provides the value that can be used for the set? size argument in the CPU_*_S() macros described below.

CPU_FREE()

Free a CPU set previously allocated by CPU_ALLOC().

The macros whose names end with "_S" are the analogs of the similarly named macros without the suffix. These macros perform the same tasks as their analogs, but operate on the dy? namically allocated CPU set(s) whose size is setsize bytes.

RETURN VALUE

CPU_ISSET() and CPU_ISSET_S() return nonzero if cpu is in set; otherwise, it returns 0.

CPU_COUNT() and CPU_COUNT_S() return the number of CPUs in set.

CPU_EQUAL() and CPU_EQUAL_S() return nonzero if the two CPU sets are equal; otherwise they return 0.

CPU_ALLOC() returns a pointer on success, or NULL on failure. (Errors are as for mal? loc(3).)

CPU_ALLOC_SIZE() returns the number of bytes required to store a CPU set of the specified cardinality.

The other functions do not return a value.

VERSIONS

The CPU_ZERO(), CPU_SET(), CPU_CLR(), and CPU_ISSET() macros were added in glibc 2.3.3.

CPU_COUNT() first appeared in glibc 2.6.

CPU_AND(), CPU_OR(), CPU_XOR(), CPU_EQUAL(), CPU_ALLOC(), CPU_ALLOC_SIZE(), CPU_FREE(), CPU_ZERO_S(), CPU_SET_S(), CPU_CLR_S(), CPU_ISSET_S(), CPU_AND_S(), CPU_OR_S(), CPU_XOR_S(), and CPU_EQUAL_S() first appeared in glibc 2.7.

CONFORMING TO

These interfaces are Linux-specific.

NOTES

To duplicate a CPU set, use memcpy(3).

Since CPU sets are bit masks allocated in units of long words, the actual number of CPUs in a dynamically allocated CPU set will be rounded up to the next multiple of sizeof(un? signed long). An application should consider the contents of these extra bits to be unde? fined.

Notwithstanding the similarity in the names, note that the constant CPU_SETSIZE indicates the number of CPUs in the cpu_set_t data type (thus, it is effectively a count of the bits in the bit mask), while the setsize argument of the CPU_*_S() macros is a size in bytes.

The data types for arguments and return values shown in the SYNOPSIS are hints what about is expected in each case. However, since these interfaces are implemented as macros, the compiler won't necessarily catch all type errors if you violate the suggestions.

BUGS

On 32-bit platforms with glibc 2.8 and earlier, CPU_ALLOC() allocates twice as much space as is required, and CPU_ALLOC_SIZE() returns a value twice as large as it should. This bug should not affect the semantics of a program, but does result in wasted memory and less efficient operation of the macros that operate on dynamically allocated CPU sets.

These bugs are fixed in glibc 2.9.

EXAMPLES

The following program demonstrates the use of some of the macros used for dynamically al? located CPU sets.

```
#define _GNU_SOURCE
#include <sched.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <assert.h>
int
main(int argc, char *argv[])
{
    cpu_set_t *cpusetp;
    size_t size;
    int num_cpus;
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <num-cpus>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    num_cpus = atoi(argv[1]);
    cpusetp = CPU_ALLOC(num_cpus);
    if (cpusetp == NULL) {
        perror("CPU_ALLOC");
        exit(EXIT_FAILURE);
    }
    size = CPU_ALLOC_SIZE(num_cpus);
    CPU_ZERO_S(size, cpusetp);
    for (int cpu = 0; cpu < num_cpus; cpu += 2)
        CPU_SET_S(cpu, size, cpusetp);
    printf("CPU_COUNT() of set:    %d\n", CPU_COUNT_S(size, cpusetp));
```

```
    CPU_FREE(cpusetp);

    exit(EXIT_SUCCESS);

}
```

SEE ALSO

sched_setaffinity(2), pthread_attr_setaffinity_np(3), pthread_setaffinity_np(3), cpuset(7)

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/.