NAME

cpuset(9) -- **CPUSET_T_INITIALIZER**, **CPUSET_FSET**, **CPU_CLR**, **CPU_COPY**, **CPU_ISSET**, **CPU_SET**, **CPU_ZERO**, **CPU_FILL**, **CPU_SETOF**, **CPU_EMPTY**, **CPU_ISFULLSET**, **CPU_FFS**, **CPU_COUNT**, **CPU_SUBSET**, **CPU_OVERLAP**, **CPU_CMP**, **CPU_OR**, **CPU_AND**, **CPU_ANDNOT**, **CPU_XOR**, **CPU_CLR_ATOMIC**, **CPU_SET_ATOMIC**, **CPU_SET_ATOMIC_ACQ**, **CPU_AND_ATOMIC**, **CPU_OR_ATOMIC**, **CPU_COPY_STORE_REL** - cpuset manipulation macros

SYNOPSIS

#include <sys/_cpuset.h>
#include <sys/cpuset.h>

CPUSET_T_INITIALIZER(*ARRAY_CONTENTS*);

CPUSET_FSET

CPU_CLR(*size_t cpu_idx*, *cpuset_t *cpuset*);

CPU_COPY(*cpuset_t *from*, *cpuset_t *to*);

bool
CPU_ISSET(*size_t cpu_idx*, *cpuset_t *cpuset*);

CPU_SET(*size_t cpu_idx*, *cpuset_t *cpuset*);

CPU_ZERO(*cpuset_t *cpuset*);

CPU_FILL(*cpuset_t *cpuset*);

CPU_SETOF(*size_t cpu_idx*, *cpuset_t *cpuset*);

bool
CPU_EMPTY(*cpuset_t *cpuset*);

bool
CPU_ISFULLSET(*cpuset_t *cpuset*);

int
CPU_FFS(*cpuset_t *cpuset*);

int

CPU_COUNT(*cpuset_t *cpuset*);

bool
CPU_SUBSET(*cpuset_t *haystack, cpuset_t *needle*);

bool
CPU_OVERLAP(*cpuset_t *cpuset1, cpuset_t *cpuset2*);

bool
CPU_CMP(*cpuset_t *cpuset1, cpuset_t *cpuset2*);

CPU_OR(*cpuset_t *dst, cpuset_t *src1, cpuset_t *src2*);

CPU_AND(*cpuset_t *dst, cpuset_t *src1, cpuset_t *src2*);

CPU_ANDNOT(*cpuset_t *dst, cpuset_t *src1, cpuset_t *src2*);

CPU_XOR(*cpuset_t *dst, cpuset_t *src1, cpuset_t *src2*);

CPU_CLR_ATOMIC(*size_t cpu_idx, cpuset_t *cpuset*);

CPU_SET_ATOMIC(*size_t cpu_idx, cpuset_t *cpuset*);

CPU_SET_ATOMIC_ACQ(*size_t cpu_idx, cpuset_t *cpuset*);

CPU_AND_ATOMIC(*cpuset_t *dst, cpuset_t *src*);

CPU_OR_ATOMIC(*cpuset_t *dst, cpuset_t *src*);

CPU_COPY_STORE_REL(*cpuset_t *from, cpuset_t *to*);

DESCRIPTION

The **cpuset(9)** family of macros provide a flexible and efficient CPU set implementation, backed by the bitset(9) macros. Each CPU is represented by a single bit. The maximum number of CPUs representable by *cpuset_t* is *CPU_SETSIZE*. Individual CPUs in cpusets are referenced with indices zero through *CPU_SETSIZE - 1*.

The **CPUSET_T_INITIALIZER**() macro allows one to initialize a *cpuset_t* with a compile time literal value.

The **CPUSET_FSET**() macro defines a compile time literal, usable by **CPUSET_T_INITIALIZER**(), representing a full cpuset (all CPUs present). For examples of **CPUSET_T_INITIALIZER**() and **CPUSET_FSET**() usage, see the *CPUSET_T_INITIALIZER EXAMPLE* section.

The **CPU_CLR**() macro removes CPU *cpu_idx* from the cpuset pointed to by *cpuset*. The **CPU_CLR_ATOMIC**() macro is identical, but the bit representing the CPU is cleared with atomic machine instructions.

The **CPU_COPY**() macro copies the contents of the cpuset *from* to the cpuset *to*. **CPU_COPY_STORE_REL**() is similar, but copies component machine words from *from* and writes them to *to* with atomic store with release semantics. (That is, if *to* is composed of multiple machine words, **CPU_COPY_STORE_REL**() performs multiple individually atomic operations.)

The **CPU_SET**() macro adds CPU *cpu_idx* to the cpuset pointed to by *cpuset*, if it is not already present. The **CPU_SET_ATOMIC**() macro is identical, but the bit representing the CPU is set with atomic machine instructions. The **CPU_SET_ATOMIC_ACQ**() macro sets the bit representing the CPU with atomic acquire semantics.

The **CPU_ISSET**() macro returns true if CPU *cpu_idx* is a member of the cpuset pointed to by *cpuset*.

The **CPU_ZERO**() macro removes all CPUs from *cpuset*.

The **CPU_FILL**() macro adds all CPUs to *cpuset*.

The **CPU_SETOF**() macro removes all CPUs in *cpuset* before adding only CPU *cpu_idx*.

The **CPU_EMPTY**() macro returns true if *cpuset* is empty.

The **CPU_ISFULLSET**() macro returns true if *cpuset* is full (the set of all CPUs).

The **CPU_FFS**() macro returns the 1-index of the first (lowest) CPU in *cpuset*, or zero if *cpuset* is empty. Like with ffs(3), to use the non-zero result of **CPU_FFS**() as a *cpu_idx* index parameter to any other **cpuset(9)** macro, you must subtract one from the result.

The **CPU_COUNT**() macro returns the total number of CPUs in *cpuset*.

The **CPU_SUBSET**() macro returns true if *needle* is a subset of *haystack*.

The **CPU_OVERLAP**() macro returns true if *cpuset1* and *cpuset2* have any common CPUs. (That is, if *cpuset1* AND *cpuset2* is not the empty set.)

The **CPU_CMP**() macro returns true if *cpuset1* is NOT equal to *cpuset2*.

The **CPU_OR**() macro adds CPUs present in *src* to *dst*. (It is the **cpuset(9)** equivalent of the scalar: *dst* |= *src*.) **CPU_OR_ATOMIC**() is similar, but sets the bits representing CPUs in the component machine words in *dst* with atomic machine instructions. (That is, if *dst* is composed of multiple machine words, **CPU_OR_ATOMIC**() performs multiple individually atomic operations.)

The **CPU_AND**() macro removes CPUs absent from *src* from *dst*. (It is the **cpuset(9)** equivalent of the scalar: *dst* &= *src*.) **CPU_AND_ATOMIC**() is similar, with the same atomic semantics as **CPU_OR_ATOMIC**().

The **CPU_ANDNOT**() macro removes CPUs in *src* from *dst*. (It is the **cpuset(9)** equivalent of the scalar: *dst* &= ~ *src*.)

CPUSET_T_INITIALIZER EXAMPLE

```
cpuset_t myset;

/* Initialize myset to filled (all CPUs) */
myset = CPUSET_T_INITIALIZER(CPUSET_FSET);

/* Initialize myset to only the lowest CPU */
myset = CPUSET_T_INITIALIZER(0x1);
```

SEE ALSO

cpuset(1), cpuset(2), bitset(9)

HISTORY

<*sys/cpuset.h*> first appeared in FreeBSD 7.1, released in January 2009, and in FreeBSD 8.0, released in November 2009.

This manual page first appeared in FreeBSD 11.0.

AUTHORS

The **cpuset(9)** macros were written by Jeff Roberson <*jeff@FreeBSD.org*>. This manual page was written by Conrad Meyer <*cem@FreeBSD.org*>.

CAVEATS

Unlike every other reference to individual set members, which are zero-indexed, **CPU_FFS**() returns a one-indexed result (or zero if the cpuset is empty).