NAME

cacheflush − flush contents of instruction and/or data cache

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/cachectl.h>

int cacheflush(void *addr***[.***nbytes***], int** *nbytes***, int** *cache***);**

Note: On some architectures, there is no glibc wrapper for this system call; see NOTES.

DESCRIPTION

cacheflush() flushes the contents of the indicated cache(s) for the user addresses in the range *addr* to *(addr+nbytes−1)*. *cache* may be one of:

ICACHE
Flush the instruction cache.

DCACHE
Write back to memory and invalidate the affected valid cache lines.

BCACHE
Same as **(ICACHE|DCACHE)**.

RETURN VALUE

cacheflush() returns 0 on success. On error, it returns −1 and sets *errno* to indicate the error.

ERRORS

EFAULT
Some or all of the address range *addr* to *(addr+nbytes−1)* is not accessible.

EINVAL
cache is not one of **ICACHE**, **DCACHE**, or **BCACHE** (but see BUGS).

STANDARDS

Historically, this system call was available on all MIPS UNIX variants including RISC/os, IRIX, Ultrix, NetBSD, OpenBSD, and FreeBSD (and also on some non-UNIX MIPS operating systems), so that the existence of this call in MIPS operating systems is a de-facto standard.

Caveat

cacheflush() should not be used in programs intended to be portable. On Linux, this call first appeared on the MIPS architecture, but nowadays, Linux provides a **cacheflush**() system call on some other architectures, but with different arguments.

NOTES

Architecture-specific variants

glibc provides a wrapper for this system call, with the prototype shown in SYNOPSIS, for the following architectures: ARC, CSKY, MIPS, and NIOS2.

On some other architectures, Linux provides this system call, with different arguments:

M68K:

```
int cacheflush(unsigned long addr, int scope, int cache,
               unsigned long len);
```

SH:

```
int cacheflush(unsigned long addr, unsigned long len, int op);
```

NDS32:

```
int cacheflush(unsigned int start, unsigned int end, int cache);
```

On the above architectures, glibc does not provide a wrapper for this system call; call it using **syscall**(2).

GCC alternative

Unless you need the finer grained control that this system call provides, you probably want to use the GCC built-in function **__builtin___clear_cache**(), which provides a portable interface across platforms supported by GCC and compatible compilers:

```
void __builtin___clear_cache(void *begin, void *end);
```

On platforms that don't require instruction cache flushes, **__builtin___clear_cache**() has no effect.

Note: On some GCC-compatible compilers, the prototype for this built-in function uses *char ** instead of *void ** for the parameters.

BUGS

Linux kernels older than Linux 2.6.11 ignore the *addr* and *nbytes* arguments, making this function fairly expensive. Therefore, the whole cache is always flushed.

This function always behaves as if **BCACHE** has been passed for the *cache* argument and does not do any error checking on the *cache* argument.