NAME

pcap_breakloop - force a pcap_dispatch() or pcap_loop() call to return

SYNOPSIS

#include <pcap/pcap.h>

void pcap_breakloop(pcap_t *);

DESCRIPTION

pcap_breakloop() sets a flag that will force **pcap_dispatch**(3) or **pcap_loop**(3) to return rather than looping; they will return the number of packets that have been processed so far, or **PCAP_ERROR_BREAK** if no packets have been processed so far. If the loop is currently blocked waiting for packets to arrive, **pcap_breakloop**() will also, on some platforms, wake up the thread that is blocked. In this version of libpcap, the only platforms on which a wakeup is caused by **pcap_breakloop**() are Linux and Windows, and the wakeup will only be caused when capturing on network interfaces; it will not be caused on other operating systems, and will not be caused on any OS when capturing on other types of devices.

This routine is safe to use inside a signal handler on UNIX or a console control handler on Windows, or in a thread other than the one in which the loop is running, as it merely sets a flag that is checked within the loop and, on some platforms, performs a signal-safe and thread-safe API call.

The flag is checked in loops reading packets from the OS - a signal by itself will not necessarily terminate those loops - as well as in loops processing a set of packets returned by the OS. **Note that if you are catching signals on UNIX systems that support restarting system calls after a signal, and calling pcap_breakloop() in the signal handler, you must specify, when catching those signals, that system calls should NOT be restarted by that signal. Otherwise, if the signal interrupted a call reading packets in a live capture, when your signal handler returns after calling pcap_breakloop(), the call will be restarted, and the loop will not terminate until more packets arrive and the call completes.**

Note also that, in a multi-threaded application, if one thread is blocked in pcap_dispatch(), pcap_loop(), pcap_next(3), or pcap_next_ex(3), a call to pcap_breakloop() in a different thread will only unblock that thread on the platforms and capture devices listed above.

If a non-zero packet buffer timeout is set on the **pcap_t**, and you are capturing on a network interface, the thread will be unblocked with the timeout expires. This is not guaranteed to happen unless at least one packet has arrived; the only platforms on which it happens are macOS, the BSDs, Solaris 11, AIX, Tru64 UNIX, and Windows.

If you want to ensure that the loop will eventually be unblocked on any other platforms, or unblocked

when capturing on a device other than a network interface, you will need to use whatever mechanism the OS provides for breaking a thread out of blocking calls in order to unblock the thread, such as thread cancellation or thread signalling in systems that support POSIX threads.

Note that if pcap_breakloop() unblocks the thread capturing packets, and you are running on a platform that supports packet buffering, there may be packets in the buffer that arrived before pcap_breakloop() were called but that weren't yet provided to libpcap, those packets will not have been processed by pcap_dispatch() or pcap_loop(). If pcap_breakloop() was called in order to terminate the capture process, then, in order to process those packets, you would have to call pcap_dispatch() one time in order to process the last batch of packets. This may block until the packet buffer timeout expires, so a non-zero packet buffer timeout must be used.

Note that **pcap_next**() and **pcap_next_ex**() will, on some platforms, loop reading packets from the OS; that loop will not necessarily be terminated by a signal, so **pcap_breakloop**() should be used to terminate packet processing even if **pcap_next**() or **pcap_next_ex**() is being used.

pcap_breakloop() does not guarantee that no further packets will be processed by **pcap_dispatch**() or **pcap_loop**() after it is called; at most one more packet might be processed.

If **PCAP_ERROR_BREAK** is returned from **pcap_dispatch**() or **pcap_loop**(), the flag is cleared, so a subsequent call will resume reading packets. If a positive number is returned, the flag is not cleared, so a subsequent call will return **PCAP_ERROR_BREAK** and clear the flag.

BACKWARD COMPATIBILITY

This function became available in libpcap release 0.8.1.

In releases prior to libpcap 1.10.0, **pcap_breakloop**() will not wake up a blocked thread on any platform.

SEE ALSO

pcap(3)