

**NAME**

**netisr** - Kernel network dispatch service

**SYNOPSIS**

```
#include <net/netisr.h>
```

*void*

```
netisr_register(const struct netisr_handler *nhp);
```

*void*

```
netisr_unregister(const struct netisr_handler *nhp);
```

*int*

```
netisr_dispatch(u_int proto, struct mbuf *m);
```

*int*

```
netisr_dispatch_src(u_int proto, uintptr_t source, struct mbuf *m);
```

*int*

```
netisr_queue(u_int proto, struct mbuf *m);
```

*int*

```
netisr_queue_src(u_int proto, uintptr_t source, struct mbuf *m);
```

*void*

```
netisr_clearqdrops(const struct netisr_handler *nhp);
```

*void*

```
netisr_getqdrops(const struct netisr_handler *nhp, uint64_t *qdrops);
```

*void*

```
netisr_getqlimit(const struct netisr_handler *nhp, u_int *qlimit);
```

*int*

```
netisr_setqlimit(const struct netisr_handler *nhp, u_int qlimit);
```

*u\_int*

```
netisr_default_flow2cpu(u_int flowid);
```

*u\_int*

```
netisr_get_cpucount(void);
```

```
u_int
```

```
netisr_get_cpuid(u_int cpunumber);
```

With optional virtual network stack support enabled via the following kernel compile option:

**options VIMAGE**

```
void
```

```
netisr_register_vnet(const struct netisr_handler *nhp);
```

```
void
```

```
netisr_unregister_vnet(const struct netisr_handler *nhp);
```

## DESCRIPTION

The **netisr** kernel interface suite allows device drivers (and other packet sources) to direct packets to protocols for directly dispatched or deferred processing. Protocol registration and work stream statistics may be monitored using `netstat(1)`.

### Protocol registration

Protocols register and unregister handlers using **netisr\_register()** and **netisr\_unregister()**, and may also manage queue limits and statistics using the **netisr\_clearqdrops()**, **netisr\_getqdrops()**, **netisr\_getqlimit()**, and **netisr\_setqlimit()**.

In case of VIMAGE kernels each virtual network stack (vnet), that is not the default base system network stack, calls **netisr\_register\_vnet()** and **netisr\_unregister\_vnet()** to enable or disable packet processing by the **netisr** for each protocol. Disabling will also purge any outstanding packet from the protocol queue.

**netisr** supports multi-processor execution of handlers, and relies on a combination of source ordering and protocol-specific ordering and work-placement policies to decide how to distribute work across one or more worker threads. Registering protocols will declare one of three policies:

**NETISR\_POLICY\_SOURCE** **netisr** should maintain source ordering without advice from the protocol. **netisr** will ignore any flow IDs present on *mbuf* headers for the purposes of work placement.

**NETISR\_POLICY\_FLOW** **netisr** should maintain flow ordering as defined by the *mbuf* header flow ID field. If the protocol implements *nh\_m2flow*, then **netisr** will query the protocol in the event that the *mbuf* doesn't have a flow ID, falling

back on source ordering.

**NETISR\_POLICY\_CPU** **netisr** will entirely delegate all work placement decisions to the protocol, querying *nh\_m2cpuid* for each packet.

Registration is declared using *struct netisr\_handler*, whose fields are defined as follows:

*const char \* nh\_name* Unique character string name of the protocol, which may be included in `sysctl(3)` MIB names, so should not contain whitespace.

*netisr\_handler\_t nh\_handler* Protocol handler function that will be invoked on each packet received for the protocol.

*netisr\_m2flow\_t nh\_m2flow* Optional protocol function to generate a flow ID and set a valid hashtype for packets that enter the **netisr** with `M_HASHTYPE_GET(m)` equal to `M_HASHTYPE_NONE`. Will be used only with `NETISR_POLICY_FLOW`.

*netisr\_m2cpuid\_t nh\_m2cpuid* Protocol function to determine what CPU a packet should be processed on. Will be used only with `NETISR_POLICY_CPU`.

*netisr\_drainedcpu\_t nh\_drainedcpu* Optional callback function that will be invoked when a per-CPU queue was drained. It will never fire for directly dispatched packets. Unless fully understood, this special-purpose function should not be used.

*u\_int nh\_proto* Protocol number used by both protocols to identify themselves to **netisr**, and by packet sources to select what handler will be used to process packets. A table of supported protocol numbers appears below. For implementation reasons, protocol numbers great than 15 are currently unsupported.

*u\_int nh\_qlimit* The maximum per-CPU queue depth for the protocol; due to internal implementation details, the effective queue depth may be as much as twice this number.

*u\_int nh\_policy* The ordering and work placement policy for the protocol, as described earlier.

**Packet source interface**

Packet sources, such as network interfaces, may request protocol processing using the **netisr\_dispatch()** and **netisr\_queue()** interfaces. Both accept a protocol number and *mbuf* argument, but while **netisr\_queue()** will always execute the protocol handler asynchronously in a deferred context, **netisr\_dispatch()** will optionally direct dispatch if permitted by global and per-protocol policy.

In order to provide additional load balancing and flow information, packet sources may also specify an opaque source identifier, which in practice might be a network interface number or socket pointer, using the **netisr\_dispatch\_src()** and **netisr\_queue\_src()** variants.

**Protocol number constants**

The follow protocol numbers are currently defined:

NETISR\_IP        IPv4

NETISR\_IGMP    IGMPv3 loopback

NETISR\_ROUTE   Routing socket loopback

NETISR\_ARP     ARP

NETISR\_IPV6    IPv6

**AUTHORS**

This manual page and the **netisr** implementation were written by Robert N. M. Watson.