NAME

SCH_QDISC − Linux Queuing Discipline Module

SYNOPSIS

```
#include <linux/config.h>
#include <linux/module.h>
#include <asm/uaccess.h>
#include <asm/system.h>
#include <asm/bitops.h>
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/jiffies.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/socket.h>
#include <linux/sockios.h>
#include <linux/in.h>
#include <linux/errno.h>
#include <linux/interrupt.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include <linux/notifier.h>
#include <linux/init.h>
#include <net/ip.h>
#include <linux/ipv6.h>
#include <net/route.h>
#include <linux/skbuff.h>
#include <net/sock.h>
#include <net/pkt_sched.h>

int register_qdisc(struct Qdisc_ops *qops);
int unregister_qdisc(struct Qdisc_ops *qops);

struct Qdisc_ops
{
        struct Qdisc_ops *next;
        struct Qdisc_class_ops *cl_ops;
        char id[IFNAMSIZ];
        int priv_size;
        int (*enqueue)(struct sk_buff *packet, struct Qdisc *qdisc);
        struct sk_buff *(*dequeue)(struct Qdisc *qdisc);
        int (*requeue)(struct sk_buff *packet, struct Qdisc *qdisc);
        unsigned int (*drop)(struct Qdisc *qdisc);
        int (*init)(struct Qdisc *qdisc, struct rtattr *opt);
        void (*reset)(struct Qdisc *qdisc);
        void (*destroy)(struct Qdisc *qdisc);
        int (*change)(struct Qdisc *qdisc, struct rtattr *opt);
        int (*dump)(struct Qdisc *qdisc, struct sk_buff *packet);
        struct module *owner;
};
```

DESCRIPTION

A queuing discipline registers itself to the kernel using **register_qdisc.** It uses **unregister_qdisc** in order to make the kernel forget about the queuing discipline. The argument **qops** must point to a static **struct Qdisc_ops** structure that has been initialised with the routines that the kernel will use to operate the queuing discipline. The members of the structure should be initialised as follows:-

next and *cl_ops* should be initialized to NULL. The kernel will assign values to these.

id should be initialised to contain a null terminated string with the name of the queuing discipline.

priv_size
must be set to the size of the main structure that will serve as an instance of the queuing discipline. The contents of this structure is defined by the queuing discipline and is private to the queuing discipline. The memory for this will be allocated by the kernel.

init must be set to the routine that will initialize the queuing discipline. *qdisc* will point to the kernel's structure for the queuing discipline instance. *qdisc_priv* can be used to obtain a pointer to the space that the kernel has allocated for the queuing discipline's private structure for the queuing discipline instance. *opt* points to a **flat** container structure used to transport the parameters of the queuing discipline from the tc user space command to the queuing discipline in the kernel. The details of this latter structure are known only to the queuing discipline and to the tc command. The macro *RCA_DATA* should be used to extract a pointer to this structure from *opt. init* must initialise an instance of the queuing discipline based on the parameters contained in *opt.*

change must either be set to NULL or be set to a routine to change the parameters for the queuing discipline instance. The arguments to *change* are the same as those for *init.*

reset must point to a routine that will reset the queuing instance given by *qdisc* to an initial state. *reset* should use packet and queue oriented routines to free all packets in the queue and free them. It should also reset all statistical data about the queue.

destroy
must point to a routine that will free any resources allocated, including packets still queued. I recommend calling *reset* to do much of this.

dump should either be set to NULL or it should be set to a routine that saves parameter information from the queuing discipline instance *qdisc* into *opt. dump* will be called when tc requests details from the kernel. It is the authors hope that more details on *dump* will be provided by experts.

enqueue
must point to a routine that will handle a request to enqueue the packet *packet* into the queuing discipline instance *qdisc. enqueue* can either enqueue the packet or drop it. Either way, it must update statistical information which depends on its actions. It should return 0 on success or NET_XMIT_DROP if the packet was dropped.

dequeue
must point to a routine that will dequeue the packet *packet* from the queuing discipline instance *qdisc.* It must update statistical information which depends on its actions. *dequeue* should return the dequeued packet or NULL if it cannot dequeue a packet because the queue is empty.

requeue
must point to a routine that will requeue the packet *packet* into the queuing discipline instance *qdisc. packet* is a dequeued packet that for some reason cannot be sent (possibly a hardware problem), so *requeue* should place *packet* at the **head** of the queue. It must update statistical information which depends on its actions. 0 (zero) should be returned to indicate success.

drop must point to a routine that will handle a request to drop the packet *packet* from the instance of the queuing discipline *queue.* It should return the length of the packet on success, or 0 on failure. *drop* must update statistical information which depends on its actions.

The following structures are referred to below:-

```
struct tc_stats
{
        __u64   bytes;
        __u32   packets;
        __u32   drops;
        ...
        __u32   qlen;
        __u32   backlog;
};

struct sk_buff_head
{
        ...
        __u32 qlen;
        ...
};

struct Qdisc
{
        ...
        struct sk_buff_head     q;
        ...
        struct tc_stats stats;
        ...
}
```

q The kernel will have created and initialised a queue for you to store your packets in. Of course, if your queuing discipline requires multiple queues or this does not suit your needs, you may well need to create your own queues. If you do so, then you must fake the length of *q* to reflect the total number of packets that you are storing. When you add or remove a packet, then you must increment or decrement the apparent length of *q.* (e.g. ++qdisc->q.qlen)

stats Some fields of *stats* must be kept up to date. As each new packet is enqueued *packets* must be incremented and *backlog* and *bytes* must be increased by the length of the packet. *backlog* in particular should always reflect the number of bytes queued. It should be decreased by the length of the packet, for example, for each packet that is dequeued and when you are explicitly requested to drop a packet (the *drop* routine is called) then *backlog* must be decreased by the length of the packet dropped. For each packet that is dropped, *drops* must be incremented.

The kernel places interrupt handling on hold while your queuing discipline is being services. This means that interrupts are transparent to your queuing discipline code and you do not have to worry about it. On the other hand, it is vitally important that none of your routines called by the kernel should take too long to execute.

SEE ALSO

tc(8)