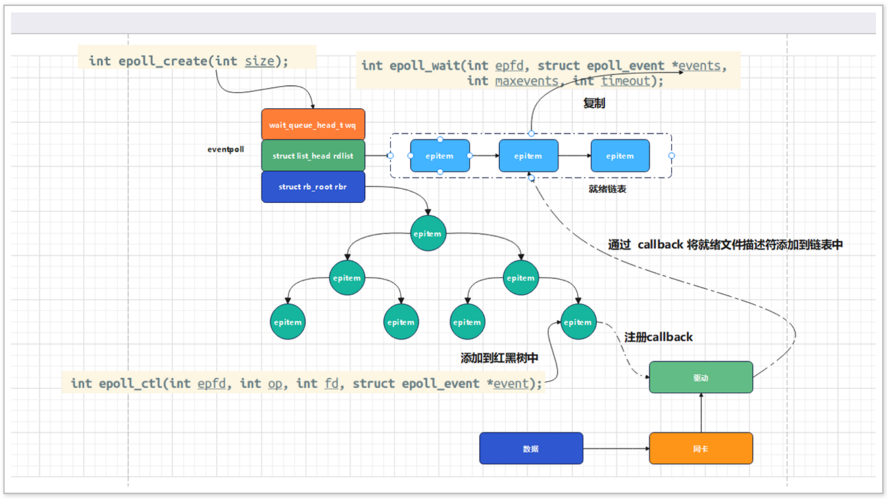
12.8 多路复用 io-epoll(三)- 原理分析_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 12.8 多路复用 io–epoll(三)– 原理分析涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。


int epoll_create(int size);
int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout);
复制
wait_queue_head_t wq
eventpoll
struct list_head rdlist
struct rb_root rbr
epitem
epitem
epitem
就绪链表
通过 callback 将就绪文件描述符添加到链表中
epitem
epitem
epitem
epitem
epitem
epitem
epitem
注册callback
添加到红黑树中
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
驱动
数据
网卡

- 上述框架中核心的数据结构 struct eventpoll，具体定义如下:

```
struct eventpoll {

    wait_queue_head_t wq;

    wait_queue_head_t poll_wait;

    struct list_head rdllist;

    struct rb_root rbr;
};
```

- 红黑树的节点对应的数据结构为 struct epitem，具体定义如下:

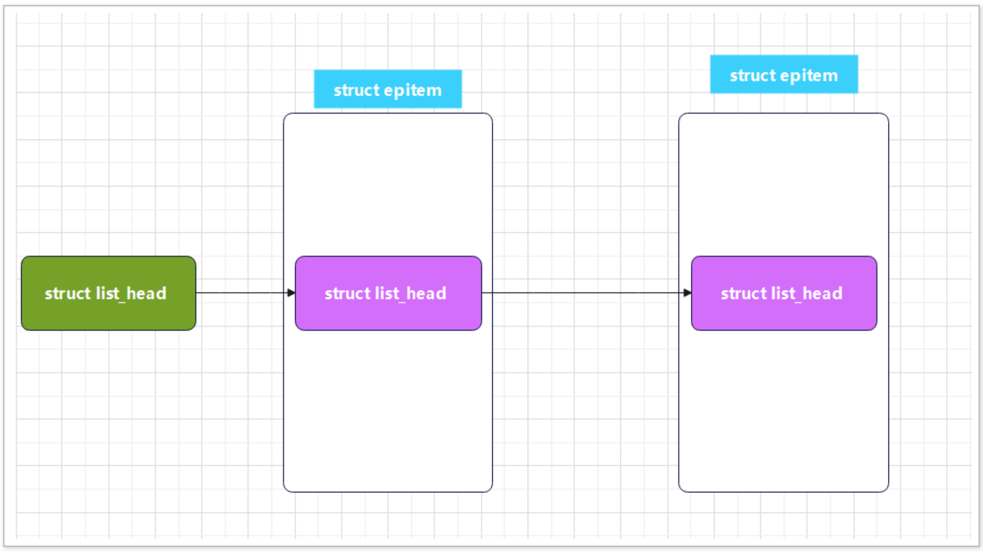
```
struct epitem {

    struct list_head rdllink;

    struct epitem *next;

    struct epoll_filefd ffd;


    struct epoll_event event;
};
```



```
SYSCALL_DEFINE1(epoll_create1, int, flags)
{
    int error, fd;
    struct eventpoll *ep = NULL;
    struct file *file;


    BUILD_BUG_ON(EPOLL_CLOEXEC != O_CLOEXEC);

    if (flags & ~EPOLL_CLOEXEC)
        return -EINVAL;

    error = ep_alloc(&ep);
    if (error < 0)
        return error;

    fd = get_unused_fd_flags(O_RDWR | (flags & O_CLOEXEC));
    if (fd < 0) {
        error = fd;
        goto out_free_ep;
    }
    file = anon_inode_getfile("[eventpoll]", &eventpoll_fops, ep,
                 O_RDWR | (flags & O_CLOEXEC));
    if (IS_ERR(file)) {
        error = PTR_ERR(file);
        goto out_free_fd;
    }
    ep->file = file;
    fd_install(fd, file);
    return fd;

out_free_fd:
    put_unused_fd(fd);
out_free_ep:
    ep_free(ep);
    return error;
}
```

- epoll_create 函数主要是为后续的操作做准备
  - 分配 struct eventpoll 的空间

  -
    ```
    struct eventpoll *ep = NULL;

    error = ep_alloc(&ep);
    ```
  - 对 eventpoll 结构进行初始化

```
 fd = get_unused_fd_flags(O_RDWR | (flags & O_CLOEXEC));
    if (fd < 0) {
        error = fd;
        goto out_free_ep;
    }
 file = anon_inode_getfile("[eventpoll]", &eventpoll_fops, ep,
                 O_RDWR | (flags & O_CLOEXEC));
    if (IS_ERR(file)) {
        error = PTR_ERR(file);
        goto out_free_fd;
```

```
    }
    ep->file = file;



SYSCALL_DEFINE4(epoll_ctl, int, epfd, int, op, int, fd,
        struct epoll_event __user *, event)
{
    int error;
    int full_check = 0;
    struct fd f, tf;
    struct eventpoll *ep;
    struct epitem *epi;
    struct epoll_event epds;
    struct eventpoll *tep = NULL;

    error = -EFAULT;
    if (ep_op_has_event(op) &&
        copy_from_user(&epds, event, sizeof(struct epoll_event)))
        goto error_return;

    epi = ep_find(ep, tf.file, fd);

    error = -EINVAL;
    switch (op) {
    case EPOLL_CTL_ADD:
        if (!epi) {
            epds.events |= POLLERR | POLLHUP;
            error = ep_insert(ep, &epds, tf.file, fd, full_check);
        } else
            error = -EEXIST;
        if (full_check)
            clear_tfile_check_list();
        break;
    case EPOLL_CTL_DEL:
        if (epi)
            error = ep_remove(ep, epi);
        else
            error = -ENOENT;
        break;
    case EPOLL_CTL_MOD:
        if (epi) {
            epds.events |= POLLERR | POLLHUP;
            error = ep_modify(ep, epi, &epds);
        } else
            error = -ENOENT;
        break;
    }


    return error;
}
```

- 在 epoll_ctl 函数中调用 ep_insert 函数来实现 EPOLL_CTL_ADD 命令

- ep_insert 函数具体如下:

```
static int ep_insert(struct eventpoll *ep, struct epoll_event *event,
             struct file *tfile, int fd, int full_check)
{
    int error, revents, pwake = 0;
    unsigned long flags;
    long user_watches;
    struct epitem *epi;
    struct ep_pqueue epq;




    init_poll_funcptr(&epq.pt, ep_ptable_queue_proc);




    ep_rbtree_insert(ep, epi);


    return error;
}
```

- ep_ptable_queue_proc 函数具体实现如下:

```
static void ep_ptable_queue_proc(struct file *file, wait_queue_head_t *whead,
                 poll_table *pt)
{
    struct epitem *epi = ep_item_from_epqueue(pt);
    struct eppoll_entry *pwq;

    if (epi->nwait >= 0 && (pwq = kmem_cache_alloc(pwq_cache, GFP_KERNEL))) {
        init_waitqueue_func_entry(&pwq->wait, ep_poll_callback);
        pwq->whead = whead;
        pwq->base = epi;
        add_wait_queue(whead, &pwq->wait);
        list_add_tail(&pwq->llink, &epi->pwqlist);
        epi->nwait++;
    } else {

        epi->nwait = -1;
    }
}
```

- 在 ep_ptable_queue_proc 中 将 ep_poll_callback 回调函数添加到等待队列，驱动最后会关联这个等待队列，调用 ep_poll_callback 函数

```
 init_waitqueue_func_entry(&pwq->wait, ep_poll_callback);
```

```
static int ep_poll_callback(wait_queue_t *wait, unsigned mode, int sync, void *key)
{
    int pwake = 0;
    unsigned long flags;
    struct epitem *epi = ep_item_from_wait(wait);
    struct eventpoll *ep = epi->ep;

    if ((unsigned long)key & POLLFREE) {
        ep_pwq_from_wait(wait)->whead = NULL;

        list_del_init(&wait->task_list);
    }

    spin_lock_irqsave(&ep->lock, flags);


    if (!(epi->event.events & ~EP_PRIVATE_BITS))
        goto out_unlock;


    if (key && !((unsigned long) key & epi->event.events))
        goto out_unlock;


    if (unlikely(ep->ovflist != EP_UNACTIVE_PTR)) {
        if (epi->next == EP_UNACTIVE_PTR) {
            epi->next = ep->ovflist;
            ep->ovflist = epi;
            if (epi->ws) {

                __pm_stay_awake(ep->ws);
            }

        }
        goto out_unlock;
    }


    if (!ep_is_linked(&epi->rdllink)) {
        list_add_tail(&epi->rdllink, &ep->rdllist);
        ep_pm_stay_awake_rcu(epi);
    }


    if (waitqueue_active(&ep->wq))
        wake_up_locked(&ep->wq);
    if (waitqueue_active(&ep->poll_wait))
        pwake++;

out_unlock:
    spin_unlock_irqrestore(&ep->lock, flags);


    if (pwake)
        ep_poll_safewake(&ep->poll_wait);
    return 1;
}
```

- ep_epoll_callback 回调函数最重要的一件事情就是将就绪的文件描述符添加到就绪链表中

```
    if (!ep_is_linked(&epi->rdllink)) {
        list_add_tail(&epi->rdllink, &ep->rdllist);
        ep_pm_stay_awake_rcu(epi);
    }


SYSCALL_DEFINE4(epoll_wait, int, epfd, struct epoll_event __user *, events,
        int, maxevents, int, timeout)
{
    int error;
    struct fd f;
    struct eventpoll *ep;


    if (maxevents <= 0 || maxevents > EP_MAX_EVENTS)
        return -EINVAL;


    if (!access_ok(VERIFY_WRITE, events, maxevents * sizeof(struct epoll_event)))
        return -EFAULT;


    f = fdget(epfd);
    if (!f.file)
        return -EBADF;


    error = -EINVAL;
    if (!is_file_epoll(f.file))
        goto error_fput;


    ep = f.file->private_data;


    error = ep_poll(ep, events, maxevents, timeout);

error_fput:
    fdput(f);
    return error;
}
```

- epoll_wait 核心函数调用的是 ep_poll 函数，具体实现如下:

```
static int ep_poll(struct eventpoll *ep, struct epoll_event __user *events,
           int maxevents, long timeout)
{
    int res = 0, eavail, timed_out = 0;
    unsigned long flags;
    long slack = 0;
    wait_queue_t wait;
    ktime_t expires, *to = NULL;

    if (timeout > 0) {
        struct timespec end_time = ep_set_mstimeout(timeout);

        slack = select_estimate_accuracy(&end_time);
        to = &expires;
        *to = timespec_to_ktime(end_time);
    } else if (timeout == 0) {

        timed_out = 1;
        spin_lock_irqsave(&ep->lock, flags);
        goto check_events;
    }

fetch_events:
    spin_lock_irqsave(&ep->lock, flags);

    if (!ep_events_available(ep)) {

        init_waitqueue_entry(&wait, current);
        __add_wait_queue_exclusive(&ep->wq, &wait);

        for (;;) {
            set_current_state(TASK_INTERRUPTIBLE);
            if (ep_events_available(ep) || timed_out)
                break;
            if (signal_pending(current)) {
                res = -EINTR;
                break;
            }

            spin_unlock_irqrestore(&ep->lock, flags);
            if (!schedule_hrtimeout_range(to, slack, HRTIMER_MODE_ABS))
                timed_out = 1;

            spin_lock_irqsave(&ep->lock, flags);
        }

        __remove_wait_queue(&ep->wq, &wait);
        __set_current_state(TASK_RUNNING);
    }
check_events:

    eavail = ep_events_available(ep);

    spin_unlock_irqrestore(&ep->lock, flags);


    if (!res && eavail &&
        !(res = ep_send_events(ep, events, maxevents)) && !timed_out)
        goto fetch_events;

    return res;
}
```

- 在上面的函数实现中，在没有文件描述符就绪时，会让进程睡眠, 等待 ep_poll_callback 来进行唤醒

```
   for (;;) {

            set_current_state(TASK_INTERRUPTIBLE);
            if (ep_events_available(ep) || timed_out)
                break;
            if (signal_pending(current)) {
                res = -EINTR;
                break;
            }
}
```