12.5 多路复用 io-poll(二) 底层原理分析_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 12.5 多路复用 io-poll(二) 底层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。

- 应用层调用 poll 函数，在内核会调用 sys_poll, sys_poll 函数定义在内核源码的 fs/select.c 文件中, 具体如下:

```
SYSCALL_DEFINE3(poll, struct pollfd __user *, ufds, unsigned int, nfds,
                int, timeout_msecs)
{
        struct timespec end_time, *to = NULL;
        int ret;

        if (timeout_msecs >= 0) {
                to = &end_time;
                poll_select_set_timeout(to, timeout_msecs / MSEC_PER_SEC,
                        NSEC_PER_MSEC * (timeout_msecs % MSEC_PER_SEC));
        }

        ret = do_sys_poll(ufds, nfds, to);
        .......
}
```

- 在上面的 poll 系统调用实现中, 最核心调用的函数 do_sys_poll 函数, 具体定义如下:

```
int do_sys_poll(struct pollfd __user *ufds, unsigned int nfds,
                struct timespec *end_time)
{
        struct poll_wqueues table;
         int err = -EFAULT, fdcount, len, size;

        long stack_pps[POLL_STACK_ALLOC/sizeof(long)];
        struct poll_list *const head = (struct poll_list *)stack_pps;
        struct poll_list *walk = head;
        unsigned long todo = nfds;

        if (nfds > rlimit(RLIMIT_NOFILE))
                return -EINVAL;

        len = min_t(unsigned int, nfds, N_STACK_PPS);
        for (;;) {
                walk->next = NULL;
                walk->len = len;
                if (!len)
                        break;

                if (copy_from_user(walk->entries, ufds + nfds-todo,
                                        sizeof(struct pollfd) * walk->len))
                        goto out_fds;

                todo -= walk->len;
                if (!todo)
                        break;

                len = min(todo, POLLFD_PER_PAGE);
                size = sizeof(struct poll_list) + sizeof(struct pollfd) * len;
                walk = walk->next = kmalloc(size, GFP_KERNEL);
                if (!walk) {
                        err = -ENOMEM;
                        goto out_fds;
                }
        }

        poll_initwait(&table);
        fdcount = do_poll(nfds, head, &table, end_time);
```

```
        poll_freewait(&table);


        for (walk = head; walk; walk = walk->next) {
                struct pollfd *fds = walk->entries;
                int j;

                for (j = 0; j < walk->len; j++, ufds++)
                        if (__put_user(fds[j].revents, &ufds->revents))
                                goto out_fds;
          }

        err = fdcount;
out_fds:
        walk = head->next;
        while (walk) {
                struct poll_list *pos = walk;
                walk = walk->next;
                kfree(pos);
        }

        return err;
}
```

- 在上面的代码中，首先进行栈空间的分配，实际为分配一个数组，具体代码如下

```
long stack_pps[POLL_STACK_ALLOC/sizeof(long)];
```

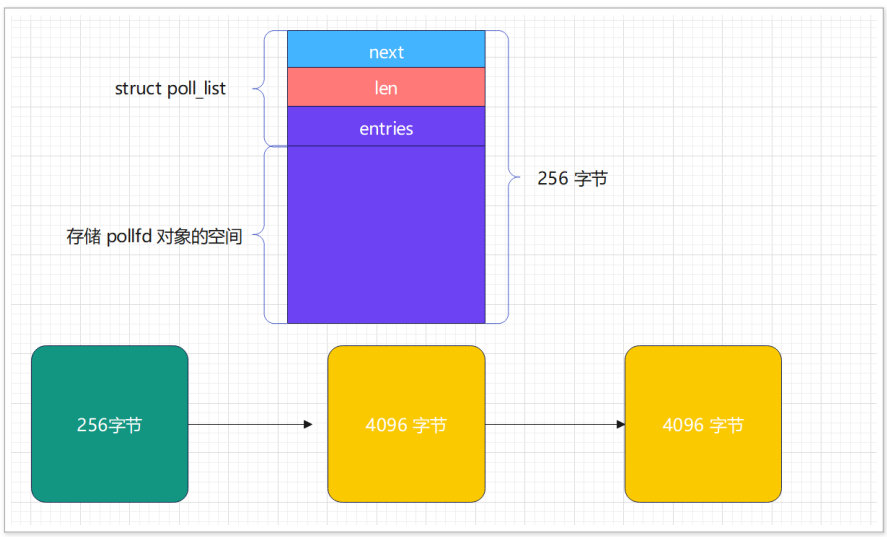
- POLL_STATCK_ALLOC 在内核中的定义大小为 256 字节, 上面的数组相当于定义了一个 256 字节的空间
- 在操作这段空间时，是基于 struct poll_list 结构体来进行，方便计算位置偏移

```
struct poll_list *const head = (struct poll_list *)stack_pps;
```

```
struct poll_list {
        struct poll_list *next;
        int len;
        struct pollfd entries[0];
};
```

- 当 256 字节的空间不够时, 则会在分配最大为 PAGE 的空间，这里会根据实际还需要多大空间进行分配

```
size = sizeof(struct poll_list) + sizeof(struct pollfd) * len;
walk = walk->next = kmalloc(size, GFP_KERNEL);
if (!walk) {
     err = -ENOMEM;
     goto out_fds;
}
```



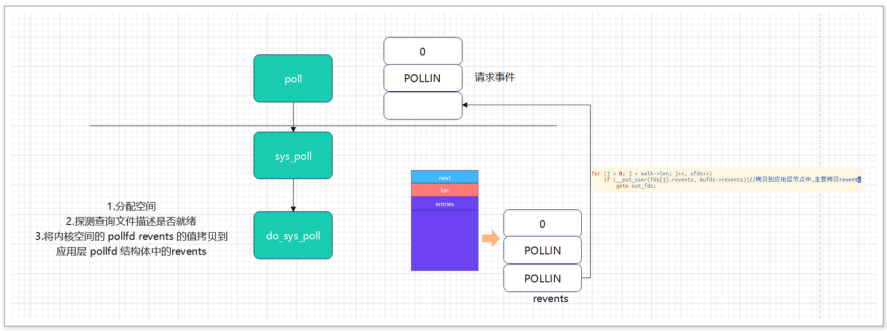
- 当内存分配好之后，就会将在用户空间的传递到内核的 struct pollfd 数组拷贝到内核的空间中

```
for (;;) {
                walk->next = NULL;
                walk->len = len;
                if (!len)
                        break;

                if (copy_from_user(walk->entries, ufds + nfds-todo,
                                        sizeof(struct pollfd) * walk->len))
                        goto out_fds;

                todo -= walk->len;
                if (!todo)
                        break;


                len = min(todo, POLLFD_PER_PAGE);
                size = sizeof(struct poll_list) + sizeof(struct pollfd) * len;

                walk = walk->next = kmalloc(size, GFP_KERNEL);

                if (!walk) {
                        err = -ENOMEM;
                        goto out_fds;
                }
        }
```

- 当用户空间的 struct pollfd 拷贝到内核空间中之后，则需要由内核进行检测是否就绪

```
static int do_poll(unsigned int nfds,  struct poll_list *list,
                   struct poll_wqueues *wait, struct timespec *end_time)
{
        poll_table* pt = &wait->pt;
        ktime_t expire, *to = NULL;
        int timed_out = 0, count = 0;
        unsigned long slack = 0;
        unsigned int busy_flag = net_busy_loop_on() ? POLL_BUSY_LOOP : 0;
        unsigned long busy_end = 0;


        if (end_time && !end_time->tv_sec && !end_time->tv_nsec) {
                pt->_qproc = NULL;
                timed_out = 1;
        }

        if (end_time && !timed_out)
                slack = select_estimate_accuracy(end_time);

        for (;;) {
                struct poll_list *walk;
                bool can_busy_loop = false;


                for (walk = list; walk != NULL; walk = walk->next) {
                        struct pollfd * pfd, * pfd_end;

                        pfd = walk->entries;
                        pfd_end = pfd + walk->len;

                        for (; pfd != pfd_end; pfd++) {

                                if (do_pollfd(pfd, pt, &can_busy_loop,
                                              busy_flag)) {
                                        count++;
                                        pt->_qproc = NULL;

                                        busy_flag = 0;
                                        can_busy_loop = false;
                                }
                        }
                }

                pt->_qproc = NULL;
                if (!count) {
                        count = wait->error;
                        if (signal_pending(current))
                                count = -EINTR;
                }
                if (count || timed_out)
                        break;
```

```
                if (can_busy_loop && !need_resched()) {
                        if (!busy_end) {
                                busy_end = busy_loop_end_time();
                                continue;
                        }
                        if (!busy_loop_timeout(busy_end))
                                continue;
                }
                busy_flag = 0;


                if (end_time && !to) {
                        expire = timespec_to_ktime(*end_time);
                        to = &expire;
                }

                if (!poll_schedule_timeout(wait, TASK_INTERRUPTIBLE, to, slack))
                        timed_out = 1;
        }
        return count;
}
```



全文完