12.7 多路复用 io-epoll(二) 控制与等待_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 12.7 多路复用 io-epoll(二) 控制与等待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。

- epoll 控制函数主要用于文件描述符集合的管理，包括增加、修改、删除等操作, 具体需要调用 epoll_ctl 函数
- 函数详细信息如下:

函数头文件 #include <sys/epoll.h>

函数原型 int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

函数参数 epfd : epoll 实例

op : epoll 操作命令字

fd : 操作的文件描述符

event : struct epoll_event 结构体对象指针

- 相关参数具体说明
  - op 为 epoll 操作命令字，具体定义如下
    - EPOLL_CTL_ADD : 在 epoll 实例中添加新的文件描述符 (相当于添加到红黑树), 并将事件链接到 fd
    - EPOLL_CTL_MOD : 更改与目标文件描述符 fd 相关联的事件
    - EPOLL_CTL_DEL : 从 epoll 实例中删除目标文件描述符 fd , 事件参数被忽略
    - 在系统中定义如下:
    - ```
      #define EPOLL_CTL_ADD 1
      #define EPOLL_CTL_DEL 2
      #define EPOLL_CTL_MOD 3
      ```
  - struct epoll_event 结构体定义如下:
  - ```
    typedef union epoll_data {
          void        *ptr;
           int          fd;
           uint32_t     u32;
           uint64_t     u64;
      } epoll_data_t;
    ```
  - ```
    struct epoll_event {
       uint32_t     events;
       epoll_data_t data;
    };
    ```
  - events : epoll 事件，事件具体定义如下:
    - EPOLLIN ： 读事件有效
    - EPOLLOUT : 写事件有效
    - EPOLLET: 将 EPOLL 设为边缘触发 (Edge Triggered) 模式
  -
  -
    ```

- epoll_data 是一个共用体，主要使用 fd 成员用于存储文件描述符
-
- 在系统中定义是一个枚举:

```
enum EPOLL_EVENTS
  {
    EPOLLIN = 0x001,
#define EPOLLIN EPOLLIN
    EPOLLPRI = 0x002,
#define EPOLLPRI EPOLLPRI
    EPOLLOUT = 0x004,
#define EPOLLOUT EPOLLOUT
    EPOLLRDNORM = 0x040,
#define EPOLLRDNORM EPOLLRDNORM
    EPOLLRDBAND = 0x080,
#define EPOLLRDBAND EPOLLRDBAND
    EPOLLWRNORM = 0x100,
#define EPOLLWRNORM EPOLLWRNORM
    EPOLLWRBAND = 0x200,
#define EPOLLWRBAND EPOLLWRBAND
    EPOLLMSG = 0x400,
#define EPOLLMSG EPOLLMSG
    EPOLLERR = 0x008,
#define EPOLLERR EPOLLERR
    EPOLLHUP = 0x010,
#define EPOLLHUP EPOLLHUP
    EPOLLRDHUP = 0x2000,
#define EPOLLRDHUP EPOLLRDHUP
    EPOLLEXCLUSIVE = 1u << 28,
#define EPOLLEXCLUSIVE EPOLLEXCLUSIVE
    EPOLLWAKEUP = 1u << 29,
#define EPOLLWAKEUP EPOLLWAKEUP
    EPOLLONESHOT = 1u << 30,
#define EPOLLONESHOT EPOLLONESHOT
    EPOLLET = 1u << 31
#define EPOLLET EPOLLET
  };
```

- 示例
- 将标准输入文件描述符添加到 epoll 实例中

- epoll 等待事件发生 (关联的文件描述符就绪), 这里调用 epoll_wait 函数

- epoll_wait 函数具体信息如下:

函数头文件 #include <sys/epoll.h>

函数原型

int epoll_wait(int epfd, struct epoll_event *events,

int maxevents, int timeout);

函数功能 等待文件描述符关联的事件发生

函数参数 epfd : epoll 实例对象

events : 存储就绪集合的数组的地址

maxevents : 就绪集合的最大值

timeout : 超时时间

函数返回值

- 成功 : 返回就绪的文件描述符数量
  - 超时返回，0
- 失败 : 返回 –1, 并设置 errno

示例

等待用户输入数据，如果没有则打印 timeout, 否则获取用户输入，并输出

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/epoll.h>

#define MAXEVENTS 10

int main(void)
{
    int epfd,ret;
    struct epoll_event ev;
    struct epoll_event ret_ev[MAXEVENTS];
    char buffer[64] = {0};

    epfd = epoll_create(1);
    if (epfd == -1){
        perror("[ERROR] epoll_create(): ");
    }

    printf("epfd = %d\n",epfd);

    ev.data.fd = 0;
    ev.events = EPOLLIN;
    ret = epoll_ctl(epfd,EPOLL_CTL_ADD,0,&ev);
    if (ret == -1){
        perror("[ERROR] epoll_ctl(): ");
        exit(EXIT_FAILURE);
    }

    for(;;){
        ret = epoll_wait(epfd,ret_ev,MAXEVENTS,1000);
        if (ret == -1){
            perror("[ERROR] epoll_wait(): ");
            exit(EXIT_FAILURE);
        }else if(ret == 0){
            printf("Timeout.\n");
        }else if (ret > 0){
            fgets(buffer,sizeof(buffer),stdin);
            printf("buffer : %s\n",buffer);
        }
    }
    return 0;
}
```

练习

使用 epoll 监听有名管道，当有名管道有数据时，读取数据并打印

全文完