□□□□□ का उपयोग □□□□□ में

epoll एक लिनक्स सिस्टम कॉल है जो कई फाइल डिस्क्रिप्टर को इवेंट्स के लिए प्रभावी रूप से मॉनिटर करने के लिए डिज़ाइन किया गया है, जैसे कि जब डेटा पढ़ने के लिए उपलब्ध है या एक सॉकेट लिखने के लिए तैयार है। यह विशेष रूप से नेटवर्क प्रोग्रामिंग में उपयोगी है, जहां एक सर्वर कई कनेक्शंस को एक साथ हैंडल करने की आवश्यकता हो सकती है। select या poll जैसे पुराने मैकेनिज्मों की तुलना में, epoll एक बड़ी संख्या में फाइल डिस्क्रिप्टर के साथ बेहतर स्केल करता है क्योंकि यह सभी डिस्क्रिप्टर को तैयार होने के लिए बार-बार स्कैन करने से बचता है।

यहाँ epoll का उपयोग करने के लिए एक स्टेप-बाय-स्टेप गाइड है, इसके बाद एक सरल उदाहरण है जिसमें एक सर्वर कनेक्शंस को स्वीकार करता है और डेटा वापस एको करता है।

---

epoll का उपयोग करने के चरण

1. एक □□□□□ इंस्टेंस बनाएं

epoll का उपयोग शुरू करने के लिए, आपको एक □□□□□ इंस्टेंस बनाना होगा, जो एक फाइल डिस्क्रिप्टर द्वारा प्रतिनिधित्व किया जाता है। epoll_create1 सिस्टम कॉल का उपयोग करें:

```
int epoll_fd = epoll_create1(0);
```

- **पैरामीटर**: बेसिक उपयोग के लिए 0 पास करें (कोई विशेष फ्लैग नहीं)। एडवांस्ड सीनारियो के लिए फ्लैग जैसे EPOLL_CLOEXEC का उपयोग किया जा सकता है।
- **रिटर्न वेल्यू**: सफलता पर एक फाइल डिस्क्रिप्टर (epoll_fd) वापस करता है, या गलती पर -1 (विस्तार के लिए errno चेक करें).

पुराना epoll_create फंक्शन समान है लेकिन एक साइज हिंट लेता है (अब अनदेखा किया जाता है), इसलिए epoll_create1 पसंद किया जाता है।

2. मॉनिटर करने के लिए फाइल डिस्क्रिप्टर जोड़ें

epoll_ctl का उपयोग करें फाइल डिस्क्रिप्टर (जैसे सॉकेट) को □□□□□ इंस्टेंस के साथ रजिस्टर करने के लिए और इवेंट्स को मॉनिटर करने के लिए जो आप चाहते हैं:

```
struct epoll_event ev;
ev.events = EPOLLIN;  //
ev.data.fd = some_fd; //
epoll_ctl(epoll_fd, EPOLL_CTL_ADD, some_fd, &ev);
```

- **पैरामीटर**:
  - epoll_fd: □□□□□ इंस्टेंस फाइल डिस्क्रिप्टर।
  - EPOLL_CTL_ADD: एक फाइल डिस्क्रिप्टर को जोड़ने का ऑपरेशन।
  - some_fd: मॉनिटर करने के लिए फाइल डिस्क्रिप्टर (जैसे एक सॉकेट).

- `&ev`: एक `struct epoll_event` पर एक पॉइंटर जो इवेंट्स और वैकल्पिक यूजर डेटा को परिभाषित करता है।

 सामान्य इवेंट्स:

- `EPOLLIN`: पढ़ने के लिए डेटा उपलब्ध है।
- `EPOLLOUT`: लिखने के लिए तैयार है।
- `EPOLLERR`: गलती हुई है।
- `EPOLLHUP`: हंग-अप (जैसे कनेक्शन बंद हो गया है).

 यूजर डेटा: `struct epoll_event` में `data` फील्ड एक फाइल डिस्क्रिप्टर (जैसे दिखाया गया है) या अन्य डेटा (जैसे एक पॉइंटर) को स्टोर कर सकता है ताकि स्रोत को पहचाना जा सके जब इवेंट्स हो।

3. इवेंट्स के लिए इंतजार करें

`epoll_wait` का उपयोग करें मॉनिटर किए गए फाइल डिस्क्रिप्टर पर इवेंट्स के लिए ब्लॉक और इंतजार करें:

```
struct epoll_event events[MAX_EVENTS];
int nfds = epoll_wait(epoll_fd, events, MAX_EVENTS, -1);
```

 पैरामीटर:

- `epoll_fd`: □□□□□ इंस्टेंस.
- `events`: ट्रिगर किए गए इवेंट्स को स्टोर करने के लिए एक एरे.
- `MAX_EVENTS`: वापस करने के लिए इवेंट्स की अधिकतम संख्या (एरे की साइज).
- `-1`: टाइमआउट मिलीसेकंड में (`-1` अनंत रूप से इंतजार करें; `0` तुरंत वापस करता है).

 रिटर्न वेल्यू: इवेंट्स के साथ फाइल डिस्क्रिप्टर की संख्या (`nfds`), या गलती पर `-1`.

4. इवेंट्स को हैंडल करें

`epoll_wait` द्वारा वापस किए गए इवेंट्स के माध्यम से लूप करें और उन्हें प्रोसेस करें:

```
for (int i = 0; i < nfds; i++) {
    if (events[i].events & EPOLLIN) {
        //          events[i].data.fd
    }
}
```

 `events` फील्ड का उपयोग बिटवाइज ऑपरेशंस (जैसे `events[i].events & EPOLLIN`) के साथ करें इवेंट टाइप को निर्धारित करने के लिए।

 `events[i].data.fd` का उपयोग करें जो फाइल डिस्क्रिप्टर इवेंट को ट्रिगर किया।

5. फाइल डिस्क्रिप्टर को मैनेज करें (वैकल्पिक)

- **हटाएं**: `epoll_ctl` का उपयोग करें `EPOLL_CTL_DEL` के साथ एक फाइल डिस्क्रिप्टर को मॉनिटर करने से रोकने के लिए:

```
epoll_ctl(epoll_fd, EPOLL_CTL_DEL, some_fd, NULL);
```

- **संशोधित करें**: `EPOLL_CTL_MOD` के साथ इवेंट्स को सेट करें:

```
ev.events = EPOLLOUT; //
epoll_ctl(epoll_fd, EPOLL_CTL_MOD, some_fd, &ev);
```

मुख्य अवधारणाएं

लेवल-ट्रिगर्ड vs. एज-ट्रिगर्ड

- **लेवल-ट्रिगर्ड (डिफ़ॉल्ट)**: `epoll` तब तक बार-बार नोटिफाई करता है जब तक कि स्थिति बनी रहे (जैसे डेटा अनपढ़ है)। अधिकांश मामलों के लिए सरल है।
- **एज-ट्रिगर्ड (`EPOLLET`)**: केवल एक बार नोटिफाई करता है जब स्थिति बदलती है (जैसे नया डेटा पहुंचता है)। सभी डेटा को पढ़ने/लिखने तक `EAGAIN` तक पहुंचने के लिए आवश्यक है; अधिक कुशल लेकिन जटिल है।
- `EPOLLET` को `ev.events` में सेट करें (जैसे `EPOLLIN | EPOLLET`) अगर एज-ट्रिगर्ड मोड का उपयोग कर रहे हैं।

नॉन-ब्लॉकिंग आई/ओ

`epoll` को अक्सर नॉन-ब्लॉकिंग फाइल डिस्क्रिप्टर के साथ जोड़ा जाता है ताकि आई/ओ ऑपरेशंस पर ब्लॉकिंग से बचा जा सके। एक सॉकेट को नॉन-ब्लॉकिंग मोड में सेट करें:

```
fcntl(fd, F_SETFL, fcntl(fd, F_GETFL) | O_NONBLOCK);
```

उदाहरण: सरल एको सर्वर

नीचे एक बुनियादी उदाहरण है जिसमें एक सर्वर `epoll` का उपयोग करता है कनेक्शंस को स्वीकार करता है और क्लाइंट्स को वापस डेटा एको करता है। यह सरलता के लिए लेवल-ट्रिगर्ड मोड का उपयोग करता है।

```
#include <sys/epoll.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <unistd.h>
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define MAX_EVENTS 10
#define PORT 8080

int main() {
    //
    int listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1) { perror("socket"); exit(1); }

    struct sockaddr_in addr = { .sin_family = AF_INET, .sin_addr.s_addr = INADDR_ANY, .sin_port = htons(PORT)
    if (bind(listen_fd, (struct sockaddr*)&addr, sizeof(addr)) == -1) { perror("bind"); exit(1); }
    if (listen(listen_fd, 5) == -1) { perror("listen"); exit(1); }

    //            -
    fcntl(listen_fd, F_SETFL, fcntl(listen_fd, F_GETFL) | O_NONBLOCK);

    //   epoll
    int epoll_fd = epoll_create1(0);
    if (epoll_fd == -1) { perror("epoll_create1"); exit(1); }

    //          epoll
    struct epoll_event ev, events[MAX_EVENTS];
    ev.events = EPOLLIN; //   -
    ev.data.fd = listen_fd;
    if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, listen_fd, &ev) == -1) { perror("epoll_ctl"); exit(1); }

    //
    while (1) {
        int nfds = epoll_wait(epoll_fd, events, MAX_EVENTS, -1);
        if (nfds == -1) { perror("epoll_wait"); exit(1); }

        for (int i = 0; i < nfds; i++) {
            int fd = events[i].data.fd;

            if (fd == listen_fd) {
                //
                int client_fd = accept(listen_fd, NULL, NULL);
```

```
                if (client_fd == -1) { perror("accept"); continue; }

                //            -
                fcntl(client_fd, F_SETFL, fcntl(client_fd, F_GETFL) | O_NONBLOCK);

                //          epoll
                ev.events = EPOLLIN;
                ev.data.fd = client_fd;
                if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd, &ev) == -1) {
                    perror("epoll_ctl: client_fd");
                    close(client_fd);
                }
            } else {
                //
                char buf[1024];
                ssize_t n = read(fd, buf, sizeof(buf));
                if (n <= 0) {
                    //      EOF:       epoll
                    if (n == -1 && (errno == EAGAIN || errno == EWOULDBLOCK)) continue;
                    close(fd);
                    epoll_ctl(epoll_fd, EPOLL_CTL_DEL, fd, NULL);
                } else {
                    //
                    write(fd, buf, n); //   :
                }
            }
        }
    }

    close(epoll_fd);
    close(listen_fd);
    return 0;
}
```

उदाहरण पर नोट्स

- **लिसनिंग सॉकेट**: नए कनेक्शंस को डिटेक्ट करने के लिए `EPOLLIN` मॉनिटर करता है।
- **क्लाइंट सॉकेट**: आने वाले डेटा को डिटेक्ट करने के लिए भी `EPOLLIN` मॉनिटर करता है।
- **संक्षेपण**: `write` पूरी तरह से पूरा होने का अनुमान लगाता है। उत्पादन में, डेटा को बफर करें और अंशिक लिखने के लिए `EPOLLOUT` का उपयोग करें।

- **गलती हैंडलिंग**: गलतियों या □□□ पर सॉकेट बंद करें और उन्हें □□□□□ से हटाएं।

सारांश

`epoll` लिनक्स में कई फाइल डिस्क्रिप्टर को हैंडल करने का एक कुशल तरीका प्रदान करता है: 1. `epoll_create1` के साथ एक इंस्टेंस बनाएं। 2. `epoll_ctl` के साथ फाइल डिस्क्रिप्टर और इवेंट्स को रजिस्टर करें। 3. `epoll_wait` के साथ इवेंट्स के लिए इंतजार करें। 4. लूप में इवेंट्स को प्रोसेस करें, मॉनिटर किए गए इवेंट्स को सेट करें या डिस्क्रिप्टर को हटाएं जैसा आवश्यक हो।

साधारण एप्लिकेशन के लिए लेवल-ट्रिगर्ड मोड सिफारिश किया जाता है। उच्च प्रदर्शन की आवश्यकता के लिए, एज-ट्रिगर्ड मोड के साथ सावधानीपूर्वक सभी उपलब्ध डेटा का हैंडलिंग करें। हमेशा `epoll` को नॉन-ब्लॉकिंग आई/ओ के साथ जोड़ें बेहतर परिणाम के लिए।