**API epoll** выполняет задачу, аналогичную задаче poll(2): отслеживает (monitoring)

несколько файловых дескрипторов, чтобы определить, возможен ли ввод-вывод для любого из них.

API epoll может использоваться как для фронт-триггер, так и для интерфейса, ориентированного на уровень-триггер (edge-triggered or a level-triggered interface), хорошо масштабируется для большого количества просматриваемых файловых дескрипторов.

Центральным понятием API epoll является экземпляр epoll — структура данных в ядре, которую с точки зрения пользовательского пространства можно рассматривать как контейнер для двух списков:

• Список интересов - *interest* list - (иногда также называемый набором epoll): набор файловых дескрипторов, в мониторинге которых процесс зарегистрировал свой интерес.

• Список готовности - *ready* list- : набор файловых дескрипторов, которые «готовы» к вводу-выводу. Готовый список представляет собой подмножество (или, точнее, набор ссылок на) интересующих вас файловых дескрипторов список. Список готовности динамически заполняется ядром как результат операций ввода-вывода с этими файловыми дескрипторами.

Для создания экземпляра epoll и управления им предусмотрены следующие системные вызовы запрос инстанции:

• **epoll\_create**(2) создает новый экземпляр epoll и возвращает файл дескриптор, относящийся к этому экземпляру. (Более новая функция epoll\_create1(2) расширяет функциональность epoll\_create(2).)

• Интерес к конкретным файловым дескрипторам затем регистрируется с помощью **epoll\_ctl**(2), которая добавляет элементы в список интересов экземпляра epoll.

• Функция **epoll\_wait**(2) ожидает событий ввода-вывода, блокируя вызывающий поток, если в данный момент нет доступных событий. (Этот системный вызов можно рассматривать как выборку элементов из готового списка экземпляра epoll .)

**Level-triggered and edge-triggered** (Уровень-триггер и фронт-триггер)

Интерфейс распределения событий epoll может вести себя как фронт-триггер (ET) и уровень-триггер (LT). Разницу между двумя механизмами можно описать следующим образом. Предположим, что происходит такой сценарий:

(1) Файловый дескриптор, представляющий сторону чтения канала (rfd), зарегистрирован в экземпляре epoll.

(2) Записывающий канал записывает 2 КБ данных на сторону записи канала.

(3) Выполняется вызов epoll\_wait(2), который вернет rfd как готовый файловый дескриптор.

(4) Читатель канала считывает 1 КБ данных из rfd.

(5) Выполняется вызов epoll\_wait(2).

Если файловый дескриптор rfd был добавлен в интерфейс epoll с использованием флага EPOLLET (запускаемый по фронту), вызов epoll\_wait(2), выполненный на шаге 5, вероятно, зависнет, несмотря на то, что доступные данные все еще присутствуют во входном буфере файла; в то же время удаленный узел может ожидать ответа на основе данных, которые он уже отправил. Причина в том, что режим запускаемый по фронту доставляет события только когда происходят изменения в контролируемом файловом дескрипторе, то есть событие будет генерироваться при каждом получении порции данных. Таким образом, на шаге 5 вызывающий может в конечном итоге ожидать некоторые данные, которые уже присутствуют во входном буфере.

В приведенном выше примере событие на rfd будет сгенерировано из-за записи, выполненной на 2, и событие потребляется на 3. Поскольку операция чтения, выполненная на 4, не потребляет все данные буфера, вызов epoll\_wait(2), выполненный на шаге 5, может быть заблокирован на неопределенный срок. Приложение, использующее флаг EPOLLET, должно использовать неблокирующие файловые дескрипторы (nonblocking file descriptors), чтобы избежать блокирующего чтения или записи, которые могут заморозить задачу, обрабатывающую несколько файловых дескрипторов. Предлагаемый способ использования epoll в качестве интерфейса, запускаемого фронтом (EPOLLET), следующий:

1. с неблокирующими файловыми дескрипторами; и (with nonblocking file descriptors; and)

(2) ожидая события только после возврата read(2) или write(2) EAGAIN. (by waiting for an event only after [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) or [write(2)](https://man7.org/linux/man-pages/man2/write.2.html) return **EAGAIN)**.

**Example for suggested usage** (Пример предлагаемого использования)

В то время как использование epoll при использовании в качестве интерфейса, запускаемого по уровню, имеет ту же семантику, что и poll(2), использование, запускаемое по фронту, требует большего разъяснения, чтобы избежать остановок в цикле событий приложения. В этом примере слушатель — это неблокирующий сокет, на котором был вызван listen(2). Функция do\_use\_fd() использует новый готовый файловый дескриптор до тех пор, пока не будет возвращен EAGAIN либо read(2), либо write(2). Управляемое событиями приложение конечного автомата должно после получения EAGAIN записать свое текущее состояние, чтобы при следующем вызове do\_use\_fd() продолжить чтение(2) или запись(2) с того места, где оно остановилось.

#define MAX\_EVENTS 10

struct epoll\_event ev, events[MAX\_EVENTS];

int listen\_sock, conn\_sock, nfds, epollfd;

/\* Code to set up listening socket, 'listen\_sock',

(socket(), bind(), listen()) omitted. \*/

epollfd = epoll\_create1(0);

if (epollfd == -1) {

perror("epoll\_create1");

exit(EXIT\_FAILURE);

}

ev.events = EPOLLIN;

ev.data.fd = listen\_sock;

if (epoll\_ctl(epollfd, EPOLL\_CTL\_ADD, listen\_sock, &ev) == -1) {

perror("epoll\_ctl: listen\_sock");

exit(EXIT\_FAILURE);

}

for (;;) {

nfds = epoll\_wait(epollfd, events, MAX\_EVENTS, -1);

if (nfds == -1) {

perror("epoll\_wait");

exit(EXIT\_FAILURE);

}

for (n = 0; n < nfds; ++n) {

if (events[n].data.fd == listen\_sock) {

conn\_sock = accept(listen\_sock,

(struct sockaddr \*) &addr, &addrlen);

if (conn\_sock == -1) {

perror("accept");

exit(EXIT\_FAILURE);

}

setnonblocking(conn\_sock);

ev.events = EPOLLIN | EPOLLET;

ev.data.fd = conn\_sock;

if (epoll\_ctl(epollfd, EPOLL\_CTL\_ADD, conn\_sock,

&ev) == -1) {

perror("epoll\_ctl: conn\_sock");

exit(EXIT\_FAILURE);

}

} else {

do\_use\_fd(events[n].data.fd);

}

}

}

При использовании в качестве интерфейса, активируемого по фронту, по соображениям производительности можно добавить дескриптор файла внутри интерфейса epoll (EPOLL\_CTL\_ADD) один раз, указав (EPOLLIN|EPOLLOUT). Это позволяет избежать постоянного переключения между EPOLLIN и EPOLLOUT, вызывая epoll\_ctl(2) с EPOLL\_CTL\_MOD.

**#include <sys/epoll.h>**

**int epoll\_ctl(int** *epfd***, int** *op***, int** *fd***,** **struct epoll\_event \*\_Nullable** *event***);**

Этот системный вызов используется для добавления, изменения или удаления записей в списке интересов экземпляра epoll(7), на который ссылается дескриптор файла epfd.

Он запрашивает выполнение операции op для целевого дескриптора файла fd. Допустимые значения аргумента op: EPOLL\_CTL\_ADD Добавление записи в список интересов дескриптора файла epoll epfd. Запись включает дескриптор файла fd, ссылку на соответствующее описание открытого файла (см. epoll(7) и open(2)) и настройки, указанные в событии.

Элемент событий структуры **epoll\_event** представляет собой битовую маску, составленную путем объединения по ИЛИ нуля или более типов событий, возвращаемых epoll\_wait(2), и входных флагов, которые влияют на его поведение, но не возвращаются.

**Доступные типы событий:**

**EPOLLIN** Связанный файл доступен для операций чтения(2).

**EPOLLOUT** Связанный файл доступен для операций записи(2).

**EPOLLRDHUP** (начиная с Linux 2.6.17) Соединение потокового сокета закрыто или завершена запись половины соединения. (Этот флаг особенно полезен для написания простого кода для обнаружения отключения однорангового узла при использовании мониторинга, запускаемого фронтом.)

**EPOLLPRI** В дескрипторе файла возникло исключительное состояние.

См. обсуждение POLLPRI в poll(2).

**EPOLLERR** В дескрипторе файла возникла ошибка. Это событие также сообщается для конца записи канала, когда конец чтения был закрыт. epoll\_wait(2) всегда будет сообщать об этом событии; нет необходимости устанавливать его в событиях при вызове epoll\_ctl(). **EPOLLHUP** Произошло зависание связанного файлового дескриптора. epoll\_wait(2) всегда будет ожидать этого события; нет необходимости устанавливать его в событиях при вызове epoll\_ctl(). Обратите внимание, что при чтении из канала, такого как канал (pipe) или потоковый сокет, это событие просто указывает на то, что одноранговый участник закрыл свой конец канала (channel). Последующие чтения из канала вернут 0 (конец файла) только после того, как все необработанные данные в канале будут использованы. **И доступны следующие входные флаги:**

**EPOLLET** Запрашивает уведомление, запускаемое фронтом, для связанного файлового дескриптора. *Поведение по умолчанию для epoll — это запуск по уровню*. См. epoll(7) для получения более подробной информации об уведомлениях, запускаемых фронтом и уровнем.

**EPOLLONESHOT** (начиная с Linux 2.6.2) Запрашивает однократное уведомление для связанного дескриптора файла. Это означает, что после события, уведомленного для дескриптора файла с помощью epoll\_wait(2), дескриптор файла отключается в списке интересов, и никакие другие события не будут сообщаться интерфейсом epoll. Пользователь должен вызвать epoll\_ctl() с EPOLL\_CTL\_MOD, чтобы перезагрузить дескриптор файла с новой маской событий.

**EPOLLWAKEUP** (начиная с Linux 3.5) Если EPOLLONESHOT и EPOLLET очищены и у процесса есть возможность CAP\_BLOCK\_SUSPEND, убедитесь, что система не переходит в режим «приостановки» или «гибернации», пока это событие ожидает обработки или обрабатывается. Событие считается "обрабатываемым" с момента, когда оно возвращается вызовом epoll\_wait(2) до следующего вызова epoll\_wait(2) в том же дескрипторе файла epoll(7), закрытия этого дескриптора файла, удаления дескриптора файла события с EPOLL\_CTL\_DEL или очистки EPOLLWAKEUP для дескриптора файла события с EPOLL\_CTL\_MOD. См. также ОШИБКИ.

**EPOLLEXCLUSIVE** (начиная с Linux 4.5) Устанавливает исключительный режим пробуждения для дескриптора файла epoll, который прикрепляется к целевому дескриптору файла, fd. Когда происходит событие пробуждения и несколько дескрипторов файлов epoll прикрепляются к одному и тому же целевому файлу с помощью EPOLLEXCLUSIVE, один или несколько дескрипторов файлов epoll получат событие с epoll\_wait(2). По умолчанию в этом сценарии (когда EPOLLEXCLUSIVE не установлен) все файловые дескрипторы epoll получают событие. Таким образом, EPOLLEXCLUSIVE полезно для избежания проблем с громоподобным стадом в определенных сценариях.

Если один и тот же файловый дескриптор находится в нескольких экземплярах epoll, некоторые с флагом EPOLLEXCLUSIVE, а другие без, то события будут предоставлены всем экземплярам epoll, которые не указали EPOLLEXCLUSIVE, и по крайней мере одному из экземпляров epoll, которые указали EPOLLEXCLUSIVE. Следующие значения могут быть указаны вместе с EPOLLEXCLUSIVE: EPOLLIN, EPOLLOUT, EPOLLWAKEUP и EPOLLET. EPOLLHUP и EPOLLERR также могут быть указаны, но это не обязательно: как обычно, эти события всегда сообщаются, если они происходят, независимо от того, указаны ли они в событиях. Попытки указать другие значения в событиях приводят к ошибке EINVAL.

EPOLLEXCLUSIVE можно использовать только в EPOLL\_CTL\_AD