Programowanie sieciowe

Obsługa operacji wejścia/wyjścia komunikacji sieciowej

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 28 maja 2019 r.

Plan wykładu

Operacje wejścia/wyjścia gniazd sieciowych

Operacje wejścia/wyjścia gniazd sieciowych (1/5)

1. Funkcje systemowe operacji wejścia:
   `read(2)`, `readv(2)`, `recv(2)`, `recvfrom(2)`, `recvmsg(2)`.
2. Funkcje systemowe operacji wyjścia:
   `write(2)`, `writev(2)`, `send(2)`, `sendto(2)`, `sendmsg(2)`.
3. Funkcje systemowe operacji dodatkowych:
   - akceptacja połączeń (przychodzących): `accept(2)`;
   - nawiązywanie połączeń (wychodzących): `connect(2)`.

Operacje wejścia/wyjścia gniazd sieciowych domyślnie są **blokujące** (w odniesieniu do niepodzielnych jednostek wymiany danych poszczególnych protokołów).

Funkcje systemowe operacji wejścia/wyjścia dla *deskryptorów przestawnych*: `pread(2)`, `preadv(2)`, `preadv2(2)`, `pwrite(2)`, `pwritev(2)`, `pwritev2(2)`.

Operacje wejścia/wyjścia gniazd sieciowych (2/5)

- Funkcje systemowe `readv(2)` i `writev(2)` podobne są do funkcji systemowych `read(2)` i `write(2)`, ale dodatkowo umożliwiają wykorzystanie więcej niż jednego bufora w pojedynczym wywołaniu funkcji (tzw. *scatter read* i *gather write*).
- Funkcje systemowe `recv(2)` i `send(2)` podobne są do funkcji systemowych `read(2)` i `write(2)`, ale dodatkowo umożliwiają określenie *flag* operacji wejścia/wyjścia (np. `MSG_DONTWAIT`).
- Funkcje systemowe `recvfrom(2)` i `sendto(2)` podobne są do funkcji systemowych `recv(2)` i `send(2)`, ale dodatkowo umożliwiają pozyskanie/określenie adresu odbiorcy/nadawcy.
- Funkcje systemowe `recvmsg(2)` i `sendmsg(2)` są najbardziej ogólne i pozwalają na zrealizowanie pozostałych (powyższych) wywołań wejścia/wyjścia.

Operacje wejścia/wyjścia gniazd sieciowych (3/5)

```
ssize_t pread(
  int fd, void *buf, size_t count, off_t offset);

ssize_t readv(
  int fd, const struct iovec *iov, int iovcnt);

ssize_t recv(
  int sockfd, void *buf, size_t len, int flags);

ssize_t recvfrom(
  int sockfd, void *buf, size_t len, int flags,
  struct sockaddr *src_addr, socklen_t *addrlen);

ssize_t recvmsg(
  int sockfd, struct msghdr *msg, int flags);
```


Operacje wejścia/wyjścia gniazd sieciowych (4/5)

```
struct iovec {
 void  *iov_base;    /* Starting address */
 size_t iov_len;     /* Number of bytes to transfer */
};

struct msghdr {
 void         *msg_name;       /* optional address */
 socklen_t     msg_namelen;    /* size of address */
 struct iovec *msg_iov;        /* scatter/gather array */
 size_t        msg_iovlen;     /* # elements in msg_iov */
 void         *msg_control;    /* ancillary data, see below */
 size_t        msg_controllen; /* ancillary data buffer len */
 int           msg_flags;      /* flags on received message */
};
```


</> Funkcja systemowa `readv(2)` — przykład

```
1  ssize_t rc;
2  char buf0[20];
3  char buf1[30];
4  char buf2[40];
5  int iovcnt;
6  struct iovec iov[3];
7
8  iov[0].iov_base = buf0;
9  iov[0].iov_len = sizeof(buf0);
10 iov[1].iov_base = buf1;
11 iov[1].iov_len = sizeof(buf1);
12 iov[2].iov_base = buf2;
13 iov[2].iov_len = sizeof(buf2);
14 ...
15 iovcnt = sizeof(iov) / sizeof(struct iovec);
16 rc = readv(fd, iov, iovcnt);
```

Operacje wejścia/wyjścia gniazd sieciowych (5/5)

```
ssize_t pwrite(
  int fd, const void *buf, size_t count, off_t offset);

ssize_t writev(
  int fd, const struct iovec *iov, int iovcnt);

ssize_t send(
  int sockfd, const void *buf, size_t len, int flags);

ssize_t sendto(
  int sockfd, const void *buf, size_t len, int flags,
  const struct sockaddr *dest_addr, socklen_t addrlen);

ssize_t sendmsg(
  int sockfd, const struct msghdr *msg, int flags);
```


Własne limity czasu dla operacji blokujących

Własne limity czasu dla operacji blokujących

Określenie własnego limitu czasu (ang. *timeout*) dla blokujących operacji wejścia/wyjścia komunikacji sieciowej możliwe jest przy użyciu następujących mechanizmów:

- funkcja systemowa `alarm(3)` oraz sygnał `SIGALRM` (lub funkcja systemowa `setitimer(2)`);
- funkcje systemowe multipleksacji wejścia/wyjścia;
- opcje `SO_SNDTIMEO` i `SO_RCVTIMEO` gniazd sieciowych.

</> Funkcja systemowa `connect(2)` z własnym limitem czasu

```
1  void callback(int signo) { return; }
2
3  int _connect(int sfd, struct sockaddr *aptr, socklen_t *lptr,
4               int tout) {
5    int c = -1;
6    sig_t sig;
7    siginterrupt(SIGALRM, 1);
8    sig = signal(SIGALRM, callback);
9    alarm(tout);
10   c = connect(sfd, aptr, lptr);
11   if (errno == EINTR) errno = ETIMEDOUT;
12   alarm(0);
13   siginterrupt(SIGALRM, 0);
14   signal(SIGALRM, sig);
15   return c;
16 }
```

</> Funkcja systemowa `read(2)` z własnym limitem czasu

```c
int readable(int fd, int tout) {
  fd_set rset;
  struct timeval tv;
  FD_ZERO(&rset);
  FD_SET(fd, &rset);
  tv.tv_sec = tout;
  tv.tv_usec = 0;
  return select(fd + 1, &rset, NULL, NULL, &tv);
}

if (readable(sfd, 1) > 0) read(sfd, buf, sizeof(buf));
```


</> Opcje limitów czasowych gniazd sieciowych

```c
int setitout(int sfd, int tout) {
  struct timeval tv;
  tv.tv_sec = tout;
  tv.tv_usec = 0;
  return setsockopt(
    sfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
}

int setotout(int sfd, int tout) {
  struct timeval tv;
  tv.tv_sec = tout;
  tv.tv_usec = 0;
  return setsockopt(
    sfd, SOL_SOCKET, SO_SNDTIMEO, &tv, sizeof(tv));
}
```


Modele obsługi operacji wejścia/wyjścia

Modele obsługi operacji wejścia/wyjścia (1/2)

Podstawowe modele obsługi operacji wejścia/wyjścia w systemach operacyjnych UNIX oraz GNU/Linux:

- blokujące operacje wejścia/wyjścia (ang. *blocking I/O*);
- nieblokujące operacje wejścia/wyjścia (ang. *nonblocking I/O*);
- multipleksacja wejścia/wyjścia (ang. *I/O multiplexing*);
- operacje wejścia/wyjścia kierowane sygnałami (ang. *signal driven I/O*; `SIGIO`);
- asynchroniczne operacje wejścia/wyjścia (ang. *asynchronous I/O*; funkcje `aio_` standardu POSIX).

Modele obsługi operacji wejścia/wyjścia (2/2)

W odniesieniu do operacji wejścia/wyjścia komunikacji sieciowej wyróżnia się dwie fazy ich działania:

1. Oczekiwanie na gotowość bufora w jądrze systemu operacyjnego.
2. Kopiowanie danych pomiędzy buforami procesu (przestrzeń użytkownika) i jądra (przestrzeń jądra).

Nieblokujące operacje wejścia/wyjścia

Ustanowienie nieblokujących operacji wejścia/wyjścia komunikacji sieciowej wymaga użycia funkcji systemowej `fcntl(2)`:

```
1  int flags = fcntl(sfd, F_GETFL, 0);
2  fcntl(sfd, F_SETFL, flags | O_NONBLOCK);
```

Dla nieblokujących gniazd przewidziany jest błąd `EWOULDBLOCK`/`EAGAIN` (ang. *would block*).

Jednorazowe użycie operacji wejścia/wyjścia komunikacji sieciowej wymaga użycia flagi `MSG_DONTWAIT`.

Operacje wejścia/wyjścia kierowane sygnałami

Ustanowienie operacji wejścia/wyjścia komunikacji sieciowej kierowanych sygnałami wymaga użycia funkcji systemowej `fcntl(2)` (lub `ioctl(2)`):

```
1  fcntl(sfd, F_SETOWN, getpid());
2  fcntl(sfd, F_SETSIG, SIGIO);
3  int flags = fcntl(sfd, F_GETFL, 0);
4  fcntl(sfd, F_SETFL, flags | O_NONBLOCK | O_ASYNC);
5  signal(SIGIO, callback);
```

Wbrew nazwie flagi, `O_ASYNC`, nie jest to model asynchroniczny.

Rozwiązanie niemal bezużyteczne dla komunikacji strumieniowej.

Asynchroniczne operacje wejścia/wyjścia (1/3)

`aio(7)` to interfejs asynchronicznych operacji wejścia/wyjścia zgodny z POSIX, który zawiera osiem funkcji z przedrostkami `aio_`.

Funkcja `aio_read(3)` rozpoczyna (w pełni) asynchroniczny odczyt danych, a `aio_write(3)` rozpoczyna (w pełni) asynchroniczny zapis danych.

```
1  struct aiocb {
2   int             aio_fildes;     /* File descriptor */
3   off_t           aio_offset;     /* File offset */
4   volatile void  *aio_buf;        /* Location of buffer */
5   size_t          aio_nbytes;     /* Length of transfer */
6   int             aio_reqprio;    /* Request priority */
7   struct sigevent aio_sigevent;   /* Notification method */
8   int             aio_lio_opcode; /* Operation to be performed*/
9  };
10 enum { LIO_READ, LIO_WRITE, LIO_NOP };
```

Asynchroniczne operacje wejścia/wyjścia (2/3)

```c
union sigval { /* Data passed with notification */
  int    sival_int;         /* Integer value */
  void *sival_ptr;          /* Pointer value */
};
struct sigevent {
  int    sigev_notify; /* Notification method */
  int    sigev_signo;  /* Notification signal */
  union sigval sigev_value;
  void  (*sigev_notify_function) (union sigval);
                       /* Function used for thread
                          notification*/
  void  *sigev_notify_attributes;
                       /* Attributes for notification thread */
  pid_t sigev_notify_thread_id;
                       /* ID of thread to signal */
};
```


Asynchroniczne operacje wejścia/wyjścia (3/3)

Metody notyfikacji o zakończonej operacji asynchronicznej (wartości pola `sigev_notify` struktury `sigevent`):

- `SIGEV_NONE`,
- `SIGEV_SIGNAL` (sygnał o numerze `sigev_signo`),
- `SIGEV_THREAD` (wywołane funkcji `sigev_notify_function`).

zob. `aio(7)`, `sigevent(7)`

</> Asynchroniczne operacje wejścia/wyjścia — przykład

```c
int sfd;
char buf[128];
struct aiocb aio;

...
memset(&aio, 0, sizeof(aio));
aio.aio_fildes = sfd;
aio.aio_offset = 0;
aio.aio_buf = buf;
aio.aio_nbytes = 128;
aio.aio_reqprio = 0;
aio.aio_sigevent.sigev_notify = SIGEV_SIGNAL;
aio.aio_sigevent.sigev_signo = SIGUSR1;
aio.aio_lio_opcode = LIO_READ;
signal(SIGUSR1, callback);
aio_read(&aio);
```


Podsumowanie

Podsumowanie

- Operacje wejścia/wyjścia gniazd sieciowych:
  `read(2)`, `readv(2)`, `recv(2)`, `recvfrom(2)`, `recvmsg(2)`;
  `write(2)`, `writev(2)`, `send(2)`, `sendto(2)`, `sendmsg(2)`;
  `accept(2)`, `connect(2)`.
- Własne limity czasu dla operacji blokujących:
  - funkcja systemowa `alarm(3)`;
  - funkcje systemowe multipleksacji wejścia/wyjścia;
  - opcje gniazd sieciowych (`SO_SNDTIMEO`, `SO_RCVTIMEO`).
- Modele obsługi operacji wejścia/wyjścia:
  - blokujące operacje wejścia/wyjścia;
  - nieblokujące operacje wejścia/wyjścia;
  - multipleksacja wejścia/wyjścia;
  - operacje wejścia/wyjścia kierowane sygnałami;
  - asynchroniczne operacje wejścia/wyjścia.

Pytania i zadania

Pytania i zadania — praca własna

1. Jakie są możliwe zastosowania funkcji systemowych `readv(2)` oraz `writev(2)`?
2. Jakie znasz rzeczywiste przypadki wykorzystania poszczególnych modeli obsługi operacji wejścia/wyjścia komunikacji sieciowej?
3. Zapoznaj się z pełną listą opcji podstawowych gniazd sieciowych — *zob.* `socket(7)`, `setsockopt(2)`.
4. Zapoznaj się z pełną listą flag dla systemowych funkcji operacji wejścia/wyjścia — *zob.* `recv(2)`, `send(2)`.

Dziękuję za uwagę!