Ch13-Advanced I/O Functions

Instructor: Limei Peng
Dept. of CSE, KNU

Outlines

- write( ) & read( )
- send( ) & recv( )
- readv( ) & writev( )

Review on write() and read()

```
#include <unistd.h>

ssize_t write(int fd, const void * buf, size_t nbytes);
```

➜ 성공 시 전달한 바이트 수, 실패 시 -1 반환

- fd — 데이터 전송대상을 나타내는 파일 디스크립터 전달.
- buf — 전송할 데이터가 저장된 버퍼의 주소 값 전달.
- nbytes — 전송할 데이터의 바이트 수 전달.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);
```

➜ 성공 시 수신한 바이트 수(단 파일의 끝을 만나면 0), 실패 시 -1 반환

- fd — 데이터 수신대상을 나타내는 파일 디스크립터 전달.
- buf — 수신한 데이터를 저장할 버퍼의 주소 값 전달.
- nbytes — 수신할 최대 바이트 수 전달.

send() and recv()

```
#include <sys/socket.h>

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

➔ 성공 시 전송된 바이트 수, 실패 시 -1 반환

- sockfd 데이터 전송 대상과의 연결을 의미하는 소켓의 파일 디스크립터 전달.
- buf 전송할 데이터를 저장하고 있는 버퍼의 주소 값 전달.
- nbytes 전송할 바이트 수 전달.
- flags 데이터 전송 시 적용할 다양한 옵션 정보 전달.

```
#include <sys/socket.h>

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

➔ 성공 시 수신한 바이트 수(단 EOF 전송 시 0), 실패 시 -1 반환

- sockfd 데이터 수신 대상과의 연결을 의미하는 소켓의 파일 디스크립터 전달.
- buf 수신된 데이터를 저장할 버퍼의 주소 값 전달.
- nbytes 수신할 수 있는 최대 바이트 수 전달.
- flags 데이터 수신 시 적용할 다양한 옵션 정보 전달.

send() and recv()

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int sockfd, void * buf, size_t nbytes, int flags);
ssize_t send(int sockfd, const void * buf, size_t nbytes, int flags);
```

flags	Description	recv	send
MSG_DONTROUTE	Bypass routing table lookup		•
MSG_DONTWAIT	Only this operation is nonblocking	•	•
MSG_OOB	Send or receive out-of-band data	•	•
MSG_PEEK	Peek at incoming message	•	
MSG_WAITALL	Wait for all the data	•	

Figure 14.6 *flags* for I/O functions.

read()/write() v.s. send()/recv()

- The major difference between them:
  - **recv()/send()** let you specify certain options for the actual operation. For example, you can set a flag (4th argument) to send out-of-band messages…
  - **read()/write()** are the "universal" file descriptor functions

옵션(Option)	의 미	send	recv
MSG_OOB	긴급 데이터(Out-of-band data)의 전송을 위한 옵션.	●	●
MSG_PEEK	입력버퍼에 수신된 데이터의 존재유무 확인을 위한 옵션.		●
MSG_DONTROUTE	데이터 전송과정에서 라우팅(Routing) 테이블을 참조하지 않을 것을 요구하는 옵션, 따라서 로컬(Local) 네트워크상에서 목적지를 찾을 때 사용되는 옵션.	●	
MSG_DONTWAIT	입출력 함수 호출과정에서 블로킹 되지 않을 것을 요구하기 위한 옵션, 즉, 넌-블로킹(Non-blocking) IO의 요구에 사용되는 옵션.	●	●
MSG_WAITALL	요청한 바이트 수에 해당하는 데이터가 전부 수신될 때까지, 호출된 함수가 반환되는 것을 막기 위한 옵션		●

MSG_PEEK & MSG_DONTWAIT

```c
/*peek_send.c*/
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>

void error_handling(char *message)
{
  fputs(message, stderr);
  fputc('\n',stderr);
  exit(1);
}

int main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in send_addr;

  if(argc!=3)
  {
   printf("Usage: %s<IP><Port>\n",argv[0]);
   exit(1);
  }

  sock = socket(PF_INET, SOCK_STREAM, 0);
  memset(&send_addr, 0, sizeof(send_addr));
  send_addr.sin_family = AF_INET;
  send_addr.sin_addr.s_addr = inet_addr(argv[1]);
  send_addr.sin_port = htons(atoi(argv[2]));

  if(connect(sock, (struct sockaddr*)&send_addr,sizeof(send_addr))==-1)
    error_handling("connectP() error!");

  write(sock, "123", strlen("123"));
  close(sock);
  return 0;
}
```

Peek_send.c (Client)

Peek_recv.c (server)

```c
/*peek_recv.c*/
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>

#define BUF_SIZE 30
void error_handling(char *message)
{
  fputs(message, stderr);
  fputc('\n',stderr);
  exit(1);
}

int main(int argc, char *argv[])
{
  int acpt_sock, recv_sock;
  struct sockaddr_in acpt_adr, recv_adr;
  int str_len, state;
  socklen_t recv_adr_sz;
  char buf[BUF_SIZE];
  if(argc!=2)
  {
    printf("Usage: %s<port>\n",argv[0]);
    exit(1);
  }

  acpt_sock = socket(PF_INET, SOCK_STREAM, 0);
  memset(&acpt_adr, 0, sizeof(acpt_adr));
  acpt_adr.sin_family = AF_INET;
  acpt_adr.sin_addr.s_addr = htonl(INADDR_ANY);
  acpt_adr.sin_port = htons(atoi(argv[1]));
```

Peek_recv.c (Cont.)

```c
  if(bind(acpt_sock, (struct sockaddr*)&acpt_adr, sizeof(acpt_adr))==-1)
     error_handling("bind() error");
  listen(acpt_sock, 5);

  recv_adr_sz = sizeof(recv_adr);
  recv_sock = accept(acpt_sock, (struct sockaddr*)&recv_adr, &recv_adr_sz);

  while(1)
  {
    str_len = recv(recv_sock, buf, sizeof(buf)-1, MSG_PEEK|MSG_DONTWAIT);
    if(str_len>0)
      break;
  }

  buf[str_len] = 0;
  printf("Buffering %d bytes: %s \n", str_len, buf);

  str_len = recv(recv_sock, buf, sizeof(buf)-1, 0);

  buf[str_len] = 0;
  printf("Read again: %s\n", buf);
  close(acpt_sock);
  close(recv_sock);
  return 0;
}
```

Receive for the 1st time

Receive for the 2nd time

Q: what will be the result if you change this option to 0?

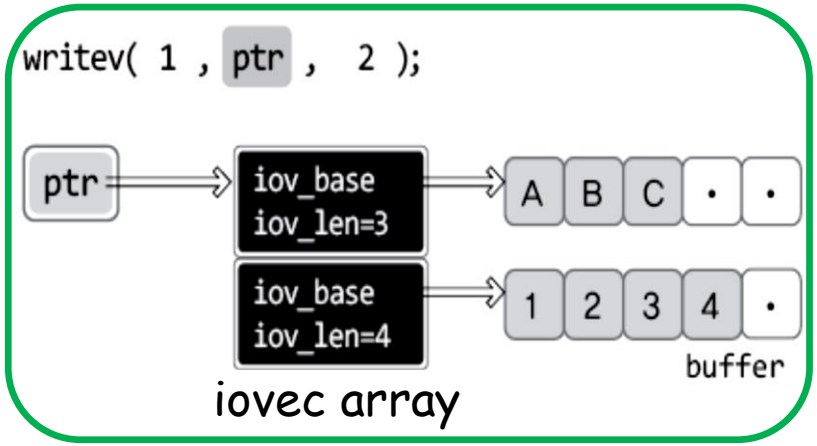
```
socket@ubuntu:~/Desktop/code$ gcc peek_recv.c -o pRecv
socket@ubuntu:~/Desktop/code$ ./pRecv 9000
Buffering 3 bytes: 123
Read again: 123
socket@ubuntu:~/Desktop/code$
```

readv( ) and writev( )

```
#include <sys/uio.h>
ssize_t writev(int filedes, const struct iovec *iov, int iovcnt);
ssize_t readv(int filedes, const struct iovec *iov, int iovcnt);
                Return: number of bytes read or written if OK, -1 on error
```

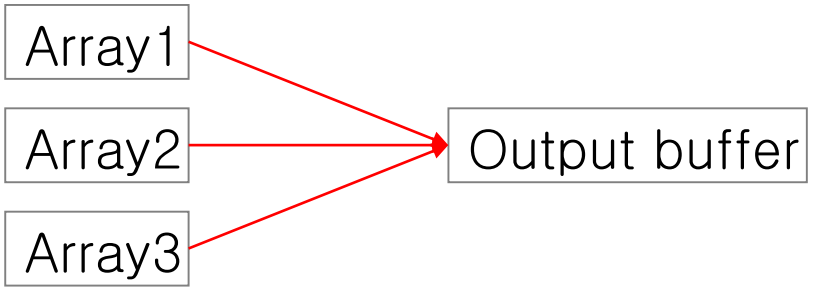
- *filedes:* File descriptor
- *iov* : /*Described below*/
- *iovcnt:* length of *iov*

```
struct iovec
{
  void * iov_base; /*starting address of buffer*/
  size_t n;        /*size of buffer*/
}
```

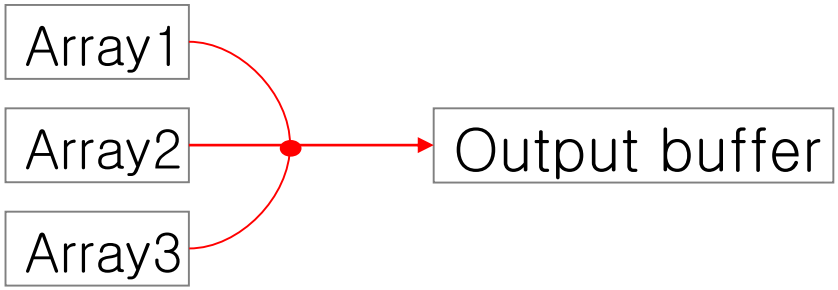


readv()/writev()

데이터를 모아서 전송(writev)

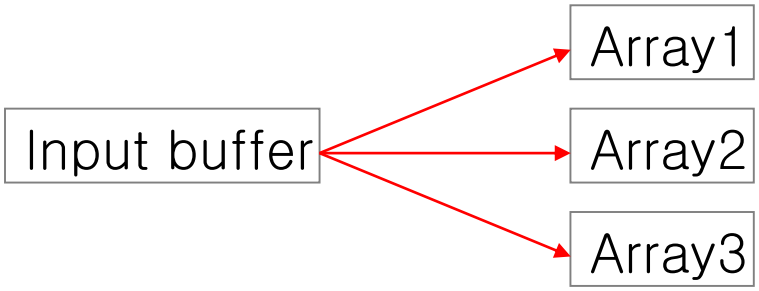


[write: should be called for three times]

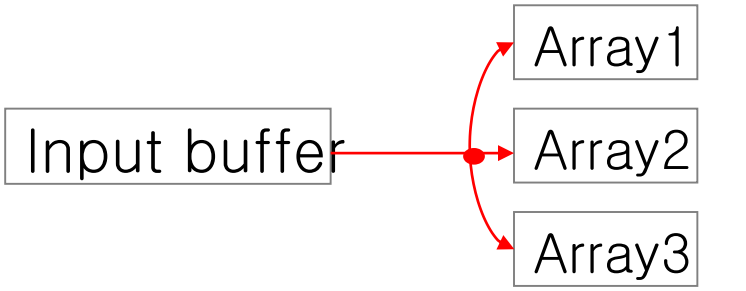


[writev need to be called for only once]

데이터를 분산 수신(readv)



[read: should be called for three times]



[readv need to be called for only once]

writev.c

```c
/*writev.c*/
#include<stdio.h>
#include<sys/uio.h>

int main(int argc, char* argv[])
{
  struct iovec vec[2];
  char buf1[] = "ABCDEFG";
  char buf2[] = "1234567";
  int str_len;

  vec[0].iov_base = buf1;
  vec[0].iov_len = 3;
  vec[1].iov_base = buf2;
  vec[1].iov_len = 4;

  str_len = writev(1, vec, 2);
  puts("");
  printf("Write bytes: %d \n", str_len);
  return 0;
}
```

```
socket@ubuntu:~/Desktop/code$ vi writev.c
socket@ubuntu:~/Desktop/code$ gcc writev.c -o wv
socket@ubuntu:~/Desktop/code$ ./wv
ABC1234
Write bytes: 7
```

readv.c

```c
/*readv.c*/
#include<stdio.h>
#include<sys/uio.h>

#define BUF_SIZE 100

int main(int argc, char *argv[])
{
  struct iovec vec[2];

  char buf1[BUF_SIZE]= {0,};
  char buf2[BUF_SIZE]= {0,};
  int str_len;

  vec[0].iov_base = buf1;
  vec[0].iov_len = 5;
  vec[1].iov_base = buf2;
  vec[1].iov_len = BUF_SIZE;

  str_len = readv(0, vec, 2);
  printf("Read bytes: %d\n", str_len);
  printf("First message: %s\n", buf1);
  printf("Second message: %s\n", buf2);
}
```

```
socket@ubuntu:~/Desktop/code$ vi readv.c
socket@ubuntu:~/Desktop/code$ gcc readv.c -o rv
socket@ubuntu:~/Desktop/code$ ./rv
I am testing readv now
Read bytes: 23
First message: I am
Second message:  testing readv now
```