BAHIRDAR INSTITUTE OF TECHNOLOGY (BIT)

DEPARTMENT OF SOFTWARE ENGINEERING

Operating System And System Programing Individual Assignment

System call of send (int sockfd ,const void *buf ,size_t len,int flags)

Done By: Michael Feleke ID: BDU1308156

Submitted To: Instructor Wondimu B.(Msc)

Submission date: 17/11/2014 E.C

Introduction to system call

What is System Call in Operating System?

A system call (syscall) is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

It is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed. This may include hardware-related services (for example, accessing a hard disk drive or accessing the device's camera), creation and execution of new processes and communication with integral kernel services such as process scheduling.

In Linux, making a system call involves transferring control from unprivileged user mode to privileged kernel mode; the details of this transfer vary from architecture to architecture. The libraries take care of collecting the system-call arguments and, if necessary, arranging those arguments in the special form necessary to make the system call.

System calls are divided into 5 categories mainly:

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communication

Scope

This article explains about the send() system call, especially about `send(int sockfd ,const void *buf ,size_t len,int flags).`

It describes about what this system call is, why this system call is used, and how this system call is implemented.

The article also describes about the list of parameters and flags of the above system call with their purpose and code implementation.

What is the send() system call?

The followings are some information about the send() system call:

DESCRIPTION

- The `send()` system call is a system call that send or pass a message to another socket. The `send()` call may be used only when the socket is in a connected state (so that the intended recipient is known).

NAME

- `send- send a message on a socket`

SYNTAX

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

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

Why is the send() system call used for?

The main function of send()function is to transmission of messages to another socket. It sends messages on a socket from another socket. It routes messages through the mail delivery system. If the delivery fails, the send command displays an error message.

How is the send() system call works?

The `send()` function shall initiate transmission of a message from the specified socket to its peer. The `send()` function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via connect()).It routes messages through the mail delivery system. If the delivery fails, the send command displays an error message.
  ```

List of parameters (The parameters are 4)

The send() function takes the following parameters:

- Socket
  - Specifies the socket file descriptor.
- Buffer
  - Points to the buffer containing the message to send.
- length
  - Specifies the length of the message in bytes.
- Flags
  - Specifies the type of message transmission.

The flag parameter

The flag parameter in send() specifies the type of message transmission. It is the bitwise OR of zero or more of the following flags :

- MSG_CONFIRM
  - ✓ Tell the link layer that forward progress happened: you got a successful reply from the other side. If the link layer doesn't get this it will regularly reprobe the neighbor (e.g., via a unicast ARP).
- MSG_DONTROUTE
  - ✓ Don't use a gateway to send out the packet, send to hosts only on directly connected networks. This is usually used only by diagnostic or routing programs. This is defined only for protocol families that route; packet sockets don't.

❖ MSG_DONTWAIT

- ✓ Enables nonblocking operation; if the operation would block, EAGAIN or EWOULDBLOCK is returned. This provides similar behavior to setting the O_NONBLOCK flag (via the fcntl(2) F_SETFL operation), but differs in that MSG_DONTWAIT is a per-call option, whereas O_NONBLOCK is a setting on the open file description.

❖ MSG_EOR

- ✓ Terminates a record (when this notion is supported, as for sockets of type SOCK_SEQPACKET).

❖ MSG_MORE

- ✓ The caller has more data to send. This flag is used with TCP sockets to obtain the same effect as the TCP_CORK socket option ,with the difference that this flag can be set on a per-call basis.

- ✓ This flag is also supported for UDP sockets, and informs the kernel to package all of the data sent in calls with this flag set into a single datagram which is transmitted only when a call is performed that does not specify this flag.

❖ MSG_NOSIGNAL

- ✓ Don't generate a SIGPIPE signal if the peer on a stream-oriented socket has closed the connection. The EPIPE error is still returned. This provides similar behavior to using sigaction(2) to ignore SIGPIPE, but, whereas MSG_NOSIGNAL is a per-call feature, ignoring SIGPIPE

sets a process attribute that affects all threads in the process.

- ❖ MSG_OOB
  - ✓ Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data.

Code implementation for MSG_CONFIRM flag

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const* argv[])
{
    int sock = 0, valread, client_fd;
    struct sockaddr_in serv_addr;
    char* hello = "Hello from client";
    char buffer[1024] = { 0 };
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
```

```
        return -1;
    }

    if ((client_fd
         = connect(sock, (struct sockaddr*)&serv_addr,
                   sizeof(serv_addr)))
        < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    send(sock, hello, strlen(hello), 0);

    printf("Hello message sent\n");
    valread = read(sock, buffer, 1024);
    printf("%s\n", buffer);

    // closing the connected socket
    close(client_fd);
    return 0;
}
```

Compiling:

```
gcc client.c -o client
gcc server.c -o server
```

Output:

```
Client: Hello message sent
Hello from server
Server: Hello from client
Hello message sent
```

Code implementation for MSG_OOB flag

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#define PORT 8080


int main(int argc, char **argv)
 {
     int     sockfd;

     if (argc != 3)
          err_quit("usage: tcpsend01 <host> <port#>");

     sockfd = Tcp_connect(argv[1], argv[2]);

     Write(sockfd, "123", 3);

     printf("wrote 3 bytes of normal data\n");

sleep(1);

Send(sockfd, "4", 1, MSG_OOB);

printf("wrote 1 byte of OOB data\n");

sleep(1);

Write(sockfd, "56", 2);

printf("wrote 2 bytes of normal data\n");

sleep(1);

Send(sockfd, "7", 1, MSG_OOB);

printf("wrote 1 byte of OOB data\n");

sleep(1);

Write(sockfd, "89", 2);

printf("wrote 2 bytes of normal data\n");

sleep(1);

     exit(0);
}
```

Output:

```
wrote 3 bytes of normal data
wrote 1 byte of OOB data
wrote 2 bytes of normal data
wrote 1 byte of OOB data
wrote 2 bytes of normal data
```

References

The Linux programming Interface by Michael Kerrisk

https://gist.github.com

https://man7.org

https://linux.die.net

https://docs.oracle.com