OPERATING SYSTEM

INDIVIDUAL ASSIGNMENT

CHROME OS

NAME	ID
MINYAMIR KELEMU	1602152

Implement System Calls

SUBMISSION DATE 16/08/2017

SUBIMETTED TO lec-wondimu

Implement System Calls (Example)

The `bind()` system call is used to associate a socket with a specific local address and port number. This is typically used by server-side applications to ensure that they listen on a specific port for incoming network connections.

Overview of `bind()` System Call:

The `bind()` function is used after creating a socket with the `socket()` system call. It associates the socket with a local address and port, which enables it to receive messages sent to that address and port.

Syntax of `bind()`:

```
c
CopyEdit
int bind(int sockfd, const struct sockaddr *addr, socklen_t
addrlen);
```

- `sockfd`: The file descriptor of the socket returned by the `socket()` call.
- `addr`: A pointer to a `sockaddr` structure that contains the address (IP address and port) to be assigned to the socket.
- `addrlen`: The length of the address structure.

Address Structures:

For IPv4, the address is usually specified in the `sockaddr_in` structure:

```
c
CopyEdit
struct sockaddr_in {
    sa_family_t    sin_family;   // Address family (AF_INET for
IPv4)
    in_port_t      sin_port;     // Port number (in network byte
order)
```

```
    struct in_addr sin_addr;     // IP address (in network byte
order)
};
```

Example of Using `bind()`:

The following code demonstrates how to create a socket, set up an address, and bind the socket to a local address and port using the `bind()` system call:

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

#define PORT 8080  // Port number to bind the socket

int main() {
    int sockfd;
    struct sockaddr_in server_addr;

    // Step 1: Create a socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Step 2: Set up the server address structure
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;           // Address
family
    server_addr.sin_addr.s_addr = INADDR_ANY;   // Any available
network interface
    server_addr.sin_port = htons(PORT);         // Port to bind
(converted to network byte order)

    // Step 3: Bind the socket to the specified address and port
    if (bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) == -1) {
        perror("Bind failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    printf("Socket successfully bound to port %d\n", PORT);

    // Additional steps can be added here, like listening for
connections

    // Close the socket
    close(sockfd);
    return 0;
}
```

Explanation of Code:

1. **Create a Socket:**
   - `socket(AF_INET, SOCK_STREAM, 0)` creates a TCP socket using IPv4.
2. **Set Up the `sockaddr_in` Structure:**
   - `server_addr.sin_family = AF_INET`: Specifies that we are using IPv4.
   - `server_addr.sin_addr.s_addr = INADDR_ANY`: Binds the socket to all available network interfaces.
   - `server_addr.sin_port = htons(PORT)`: Converts the port number `PORT` to network byte order using `htons()`.
3. **Bind the Socket:**
   - `bind()` is called to bind the socket to the address stored in `server_addr`. If successful, the socket will be bound to port 8080 (or whichever port is specified).
4. **Error Handling:**
   - The code checks for errors in socket creation and binding. If either fails, it prints an error message and exits.

Return Value:

- **On success:** `bind()` returns 0.
- **On failure:** `bind()` returns -1 and sets `errno` to indicate the error.

Common Errors:

- `EADDRINUSE`: The address is already in use (another process is already bound to the same port).
- `EACCES`: The user does not have permission to bind to the specified address or port (e.g., trying to bind to a port < 1024 as a non-root user).
- `EINVAL`: The provided address is invalid or incomplete.

Conclusion:

The `bind()` system call is essential for servers and applications that need to listen on a specific port. It enables the server to define the local address and port to which the socket is bound, ensuring that the server can receive data sent to that address and port.