Lab Preparation: File Systems and Network Access in Linux

Objectives

- Implement a TCP server to receive text via network and log it.
- Transfer data using message queues and shared memory.
- Analyze network traffic using Wireshark.

Task 3.0: TCP Server via Internet Superdaemon

Server Program (C)

```c
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#define LOG_FILE "logfile.txt"  // Define the log file name
#define BUFFER_SIZE 1024        // Define the buffer size for reading data

int main() {
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;

    // Open log file for appending
    int log_fd = open(LOG_FILE, O_WRONLY | O_APPEND | O_CREAT, 0644);
    if (log_fd < 0) {
        perror("Failed to open log file");
        return 1;
    }

    // Read data from stdin (provided by inetd/xinetd)
    while ((bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0)
   {
        // Write data to log file
        if (write(log_fd, buffer, bytes_read) != bytes_read) {
```

```
            perror("Failed to write to log file");
            close(log_fd);
            return 1;
        }
    }

    if (bytes_read < 0) {
        perror("Failed to read from socket");
    }

    close(log_fd);  // Close the log file
    return 0;
}
```

Listing 1: C program for TCP server

inetd/xinetd Configuration (Already done in lab, no need to change)

inetd

1. Add the following line to `/etc/inetd.conf`:

```
myservice stream tcp nowait nobody /path/to/tcp_server tcp_server
```

2. Add the service to `/etc/services`:

```
myservice 12345/tcp
```

3. Restart `inetd` to apply the changes:

```
sudo service inetd restart
```

xinetd

1. Create a configuration file `/etc/xinetd.d/myservice` with the following content:

```
service myservice
{
    type        = UNLISTED
    port        = 12345
    socket_type = stream
    wait        = no
    user        = nobody
    server      = /path/to/tcp_server
    log_on_failure += USERID
    disable     = no
}
```

2. Restart `xinetd` to apply the changes:

```
sudo service xinetd restart
```

Testing the Server

1. Use `telnet` to contact the server:

```
telnet localhost 12345
```

2. Or use `nc` (netcat):

```
echo "Hello, Server" | nc localhost 12345
```

Observing System Behavior

1. Check running processes:

```
ps aux | grep tcp_server
```

2. Check network status:

```
netstat -an | grep 12345
```

Using Wireshark

Wireshark is a network protocol analyzer that captures and displays data packets traveling through the network. It helps in analyzing the communication between devices on a network.

1. Install Wireshark:

```
sudo apt-get install wireshark
```

2. Start Wireshark and capture packets on the relevant network interface.
3. Use a filter to focus on the traffic on port 12345:

```
tcp.port == 12345
```

4. Observe the connection and disconnection phases. You can save the captured packets to a log file for further analysis.

Task 3.1: Message Queues

Sender Program

```c
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define QUEUE_KEY 12345      // Define the key for the message queue
#define MESSAGE_TYPE 1       // Define the message type
```

```c
// Define the message structure
struct message {
    long msg_type;
    char msg_text[100];
};

int main() {
    // Get the message queue ID
    int msg_id = msgget(QUEUE_KEY, 0666 | IPC_CREAT);
    if (msg_id < 0) {
        perror("msgget");
        exit(1);
    }

    // Prepare the message
    struct message msg;
    msg.msg_type = MESSAGE_TYPE;
    strcpy(msg.msg_text, "Hello from sender!");

    // Send the message
    if (msgsnd(msg_id, &msg, sizeof(msg.msg_text), 0) < 0) {
        perror("msgsnd");
        exit(1);
    }

    printf("Message sent: %s\n", msg.msg_text);
    return 0;
}
```

Listing 2: C program for message queue sender

Receiver Program

```c
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define QUEUE_KEY 12345      // Define the key for the message queue
#define MESSAGE_TYPE 1       // Define the message type

// Define the message structure
struct message {
    long msg_type;
    char msg_text[100];
};

int main() {
    // Get the message queue ID
    int msg_id = msgget(QUEUE_KEY, 0666 | IPC_CREAT);
    if (msg_id < 0) {
```

```
        perror("msgget");
        exit(1);
    }

    // Receive the message
    struct message msg;
    if (msgrcv(msg_id, &msg, sizeof(msg.msg_text), MESSAGE_TYPE, 0) < 0) {
        perror("msgrcv");
        exit(1);
    }

    printf("Message received: %s\n", msg.msg_text);
    return 0;
}
```

Listing 3: C program for message queue receiver

Compile and run the programs:

```
gcc -o sender sender.c
gcc -o receiver receiver.c

./receiver &
./sender
```

Task 3.2: Shared Memory

Writer Program

```
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

#define SHM_KEY 12345        // Define the key for the shared memory
#define SHM_SIZE 1024        // Define the size of the shared memory

int main() {
    // Get the shared memory ID
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666 | IPC_CREAT);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    // Attach to the shared memory
    char *shm_ptr = (char *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *)-1) {
```

```
        perror("shmat");
        exit(1);
    }

    // Write to the shared memory
    strcpy(shm_ptr, "Hello from writer!");
    printf("Data written to shared memory: %s\n", shm_ptr);

    // Detach from the shared memory
    shmdt(shm_ptr);
    return 0;
}
```

Listing 4: C program for shared memory writer

Reader Program

```
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

#define SHM_KEY 12345        // Define the key for the shared memory
#define SHM_SIZE 1024        // Define the size of the shared memory

int main() {
    // Get the shared memory ID
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    // Attach to the shared memory
    char *shm_ptr = (char *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *)-1) {
        perror("shmat");
        exit(1);
    }

    // Read from the shared memory
    printf("Data read from shared memory: %s\n", shm_ptr);

    // Detach from and remove the shared memory
    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, NULL);
    return 0;
}
```

Listing 5: C program for shared memory reader

Compile and run the programs:

```
gcc -o writer writer.c
gcc -o reader reader.c

./writer
./reader
```

Measure Data Transfer Rates

Modify the writer program to measure data transfer rates:

```c
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <time.h>

#define SHM_KEY 12345        // Define the key for the shared memory
#define SHM_SIZE 1024        // Define the size of the shared memory

int main() {
    // Get the shared memory ID
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666 | IPC_CREAT);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    // Attach to the shared memory
    char *shm_ptr = (char *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *)-1) {
        perror("shmat");
        exit(1);
    }

    // Define the data size for the test
    size_t data_size = 256; // Modify this to test different sizes
    memset(shm_ptr, 'A', data_size);
    shm_ptr[data_size] = '\0';

    // Measure the data transfer rate
    clock_t start = clock();
    for (int i = 0; i < 100000; ++i) {
        memcpy(shm_ptr, shm_ptr, data_size);
    }
    clock_t end = clock();

    // Calculate and print the transfer rate
    double time_spent = (double)(end - start) / CLOCKS_PER_SEC;
```

```
    printf("Data transfer rate for %zu bytes: %f MB/s\n", data_size, (
   data_size * 100000.0) / (1024 * 1024 * time_spent));

    // Detach from the shared memory
    shmdt(shm_ptr);
    return 0;
}
```

Listing 6: C program for measuring data transfer rates

Run the modified programs on both the lab PC and the BeagleBoard to compare the results.