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

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 with Parameter Variation

```c
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.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(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <message_size> <num_messages> <delay>\n
   ", argv[0]);
        exit(1);
    }

    int message_size = atoi(argv[1]);
    int num_messages = atoi(argv[2]);
    int delay = atoi(argv[3]);

    if (message_size <= 0 || message_size > 100) {
        fprintf(stderr, "Message size must be between 1 and 100\n");
        exit(1);
    }
    if (num_messages <= 0) {
        fprintf(stderr, "Number of messages must be greater than 0\n");
        exit(1);
    }
    if (delay < 0) {
        fprintf(stderr, "Delay must be 0 or greater\n");
        exit(1);
    }

    int msg_id = msgget(QUEUE_KEY, 0666 | IPC_CREAT);
    if (msg_id < 0) {
        perror("msgget");
        exit(1);
    }

    struct message msg;
    msg.msg_type = MESSAGE_TYPE;
    memset(msg.msg_text, 'A', message_size);
    msg.msg_text[message_size] = '\0';

    for (int i = 0; i < num_messages; ++i) {
        if (msgsnd(msg_id, &msg, message_size + 1, 0) < 0) {
            perror("msgsnd");
            exit(1);
        }
        printf("Message sent: %s\n", msg.msg_text);
        sleep(delay);
    }

    return 0;
}
```

Listing 2: C program for message queue sender

Receiver Program with Parameter Variation

```
// Include necessary headers
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.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(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <num_messages> <delay>\n", argv[0]);
        exit(1);
    }

    int num_messages = atoi(argv[1]);
    int delay = atoi(argv[2]);

    if (num_messages <= 0) {
        fprintf(stderr, "Number of messages must be greater than 0\n");
        exit(1);
    }
    if (delay < 0) {
        fprintf(stderr, "Delay must be 0 or greater\n");
        exit(1);
    }

    int msg_id = msgget(QUEUE_KEY, 0666 | IPC_CREAT);
    if (msg_id < 0) {
        perror("msgget");
        exit(1);
    }

    struct message msg;
    for (int i = 0; i < num_messages; ++i) {
        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);
        sleep(delay);
    }

    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 10 2 &
./sender 10 10 1
./sender 50 10 1
./sender 100 10 1
```

Task 3.2: Shared Memory

Writer Program with Handshake

```
// 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 <unistd.h>

#define SHM_KEY 12345
#define SHM_SIZE 1024

typedef struct {
    int data_ready;  // Writer sets this to 1 when data is ready
    int data_ack;    // Reader sets this to 1 when data is read
    char data[SHM_SIZE - 2 * sizeof(int)];  // Data buffer
} shared_memory_t;

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666 | IPC_CREAT);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    shared_memory_t *shm_ptr = (shared_memory_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (shared_memory_t *)-1) {
        perror("shmat");
```

```
        exit(1);
    }

    // Initialize the flags
    shm_ptr->data_ready = 0;
    shm_ptr->data_ack = 0;

    // Write to shared memory
    strcpy(shm_ptr->data, "Hello from writer!");
    shm_ptr->data_ready = 1;  // Set the data ready flag

    // Wait for the reader to acknowledge reading the data
    while (shm_ptr->data_ack == 0) {
        sleep(1);
    }

    // Reset the acknowledgment flag
    shm_ptr->data_ack = 0;

    shmdt(shm_ptr);
    return 0;
}
```

Listing 4: C program for shared memory writer

Reader Program with Handshake

```
// 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 <unistd.h>

#define SHM_KEY 12345
#define SHM_SIZE 1024

typedef struct {
    int data_ready;  // Writer sets this to 1 when data is ready
    int data_ack;    // Reader sets this to 1 when data is read
    char data[SHM_SIZE - 2 * sizeof(int)];  // Data buffer
} shared_memory_t;

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }
```

```
    shared_memory_t *shm_ptr = (shared_memory_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (shared_memory_t *)-1) {
        perror("shmat");
        exit(1);
    }

    // Wait for the writer to set the data ready flag
    while (shm_ptr->data_ready == 0) {
        sleep(1);
    }

    // Read from shared memory
    printf("Data read from shared memory: %s\n", shm_ptr->data);
    shm_ptr->data_ready = 0;  // Clear the data ready flag
    shm_ptr->data_ack = 1;    // Set the acknowledgment flag

    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, NULL);
    return 0;
}
```

Listing 5: C program for shared memory reader

Writer Program for Measuring Data Transfer Rates

```
// 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 SHM_SIZE 1024

typedef struct {
    int data_ready;  // Writer sets this to 1 when data is ready
    int data_ack;    // Reader sets this to 1 when data is read
    char data[SHM_SIZE - 2 * sizeof(int)];  // Data buffer
} shared_memory_t;

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <data_size>\n", argv[0]);
        exit(1);
    }

    int data_size = atoi(argv[1]);
    if (data_size <= 0 || data_size > (SHM_SIZE - 2 * sizeof(int))) {
        fprintf(stderr, "Data size must be between 1 and %d\n", (SHM_SIZE
   - 2 * sizeof(int)));
```

```
        exit(1);
    }

    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666 | IPC_CREAT);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    shared_memory_t *shm_ptr = (shared_memory_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (shared_memory_t *)-1) {
        perror("shmat");
        exit(1);
    }

    // Fill data with a specific size
    memset(shm_ptr->data, 'A', data_size);
    shm_ptr->data[data_size] = '\0';  // Ensure null termination

    // Initialize the flags
    shm_ptr->data_ready = 0;
    shm_ptr->data_ack = 0;

    clock_t start = clock();
    for (int i = 0; i < 100000; ++i) {
        // Write to shared memory and set the flag
        shm_ptr->data_ready = 1;  // Set the data ready flag

        // Wait for the reader to acknowledge reading the data
        while (shm_ptr->data_ack == 0) {
            // Busy wait
        }

        // Reset the acknowledgment flag
        shm_ptr->data_ack = 0;
    }
    clock_t end = clock();

    double time_spent = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Data transfer rate for %d bytes: %f MB/s\n", data_size, (
   data_size * 100000.0) / (1024 * 1024 * time_spent));

    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, NULL);
    return 0;
}
```

Listing 6: C program for measuring data transfer rates

Reader Program for Measuring Data Transfer Rates

```
// Include necessary headers
```

```c
#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 SHM_SIZE 1024

typedef struct {
    int data_ready;  // Writer sets this to 1 when data is ready
    int data_ack;    // Reader sets this to 1 when data is read
    char data[SHM_SIZE - 2 * sizeof(int)];  // Data buffer
} shared_memory_t;

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    shared_memory_t *shm_ptr = (shared_memory_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (shared_memory_t *)-1) {
        perror("shmat");
        exit(1);
    }

    for (int i = 0; i < 100000; ++i) {
        // Wait for the writer to set the data ready flag
        while (shm_ptr->data_ready == 0) {
            // Busy wait
        }

        // Read from shared memory and clear the flag
        shm_ptr->data_ready = 0;  // Clear the data ready flag
        shm_ptr->data_ack = 1;    // Set the acknowledgment flag
    }

    shmdt(shm_ptr);
    return 0;
}
```

Listing 7: C program for measuring data transfer rates

Compile and run the programs:

```
gcc -o writer_measure writer_measure.c
gcc -o reader reader.c

./reader &
./writer_measure 256  # Measure with data block size of 256 bytes
```

```
./writer_measure 512  # Measure with data block size of 512 bytes
./writer_measure 1024 # Measure with data block size of 1024 bytes
```