

CHATGPT LINKS

1. <https://chatgpt.com/share/673cd613-6a74-8002-bec9-f02e05bc2565>
2. <https://chatgpt.com/share/673cd5d7-5470-800b-8cea-11084c0464cd>
3. <https://chatgpt.com/share/673cd6d8-31e4-800b-b20b-4e89c2fe6b8f>
4. <https://chatgpt.com/share/673cd6f3-f580-800b-92de-e6d31ce8d050>
5. <https://chatgpt.com/share/673cd709-b2fc-800b-8624-f1ea044dc6c5>
6. <https://chatgpt.com/share/673cd723-ea18-800b-bf10-80a7ee4c0691>
7. <https://chatgpt.com/share/673cd735-6658-800b-8f5d-4794cea99079>
8. <https://chatgpt.com/share/673cd872-9ac4-8006-90a2-bbf00c971e16>
9. <https://chatgpt.com/share/6736f0b3-fda8-8006-bcf4-a8dc96474fd1>
10. <https://chatgpt.com/share/673d18f4-c454-800e-9365-3b345bc10d1e>
11. <https://chatgpt.com/share/673d1913-54d8-800e-bdfb-f6e5c911c828>
12. <https://chatgpt.com/share/673d1922-8f7c-800e-ac80-d032e1269100>
13. <https://chatgpt.com/share/673d1c61-e4d8-800e-9ac9-d96246bdcb76>
14. <https://chatgpt.com/share/673d1c73-7a1c-800e-b154-41a72900ee82>

Other AI sources

- 1.

nm.h Current File nm.c File nm_requests.c File

@nm.c @nm_requests.c debug the given codes to achieve proper backup of a server to 2 more servers

To debug the provided code for achieving proper backup of a server to two more servers, we need to ensure that the backup logic is correctly implemented in both `nm.c` and `nm_requests.c`. Below are the necessary changes and improvements to ensure that the backup functionality works as intended.

Changes in `nm.c`

1. Fix the `create_backup_directory` function: Ensure that the acknowledgment message is correctly parsed and that the backup path is properly assigned.
2. Ensure that the backup logic in `backup_storage_server` correctly handles the backup process.

Here are the relevant changes:

nm.c

↶ Ask

📋 Copy

▶ Apply

```
// ... existing code ...
```

```
void create_backup_directory(int server_index, char backup_folder[], char** ans_path) {
    // Create a command to create the backup directory
    char create_backup_cmd[BUFFER_SIZE];
    snprintf(create_backup_cmd, sizeof(create_backup_cmd), "BACKUP %s", backup_folder);
    // Send the command to the specified storage server
    send(storage_servers[server_index].sock, create_backup_cmd, strlen(create_backup_cmd), 0);
    char ack_buffer[BUFFER_SIZE];
    int bytes_received = read(storage_servers[server_index].sock, ack_buffer, sizeof(ack_buffer));
    if (bytes_received > 0) {
        ack_buffer[bytes_received] = '\0'; // Null-terminate the received string
        // Ensure the acknowledgment is parsed correctly
        char status[10]; // Adjust size as needed
        if (sscanf(ack_buffer, "%s %s", status, *ans_path) == 2) {
            printf("%s\n", ack_buffer);
            return;
        } else {
            fprintf(stderr, "Failed to parse acknowledgment message: %s\n", ack_buffer);
            return;
        }
    } else {
        printf("No acknowledgment received from server %d\n", server_index);
    }
}
```

C nm.c

↶ Ask

📄 Copy

▶ Apply

```
// ... existing code ...

void backup_storage_server(int new_server_index) {
    if (storage_server_count >= 2) {
        // Get the current storage server
        StorageServer *current_server = &storage_servers[new_server_index];

        // Get the indices of the backup servers
        int backup_server_index_1 = new_server_index - 1;
        int backup_server_index_2 = new_server_index - 2;

        // Define backup folder names
        char backup_folder_1[BUFFER_SIZE];
        char backup_folder_2[BUFFER_SIZE];
        snprintf(backup_folder_1, sizeof(backup_folder_1), "backup_%d_%d", new_server_index, backup_server_index_1);
        snprintf(backup_folder_2, sizeof(backup_folder_2), "backup_%d_%d", new_server_index, backup_server_index_2);

        // Create backup directories in the backup servers
        create_backup_directory(backup_server_index_1, backup_folder_1, &current_server);
        create_backup_directory(backup_server_index_2, backup_folder_2, &current_server);

        // Backup to the previous two storage servers
        for (int i = 0; i < current_server->path_count; i++) {
            char *src_path = current_server->accessible_paths[i];
            // Backup to server 1
            char comm1[MAX_REQ_LENGTH];
            snprintf(comm1, sizeof(comm1), "BACKUP_COPY %s %s %s %d",
                    src_path, current_server->path_backup1, storage_servers[backup_server_index_1].client_port,
                    storage_servers[backup_server_index_1].client_port);
            send(current_server->sock, comm1, strlen(comm1), 0);
            // Wait for acknowledgment from backup server 1
            // ... (handle acknowledgment as needed)

            // Backup to server 2
            char comm2[MAX_REQ_LENGTH];
            snprintf(comm2, sizeof(comm2), "BACKUP_COPY %s %s %s %d",
                    src_path, current_server->path_backup2, storage_servers[backup_server_index_2].client_port,
                    storage_servers[backup_server_index_2].client_port);
            send(current_server->sock, comm2, strlen(comm2), 0);
            // Wait for acknowledgment from backup server 2
            // ... (handle acknowledgment as needed)
        }
    }
}
```

C nm.c

↶ Ask

📄 Copy

▶ Apply

Changes in nm_requests.c

1. Ensure that the `handle_copy_backup` function correctly handles the backup process.
2. Make sure that the acknowledgment messages are sent back to the naming server correctly.

Here are the relevant changes:

C nm_requests.c

↶ Ask

📄 Copy

▶ Apply

```
// ... existing code ...

void handle_copy_backup(int nm_sock, char *src_path, char *dest_dir, const char *dest_i
    struct stat src_stat;
    if (stat(src_path, &src_stat) == -1) {
        send_ack_to_nm(nm_sock, "COPY_FAILED", src_path);
        perror("Invalid source path");
        return;
    }

    char dest_path[MAX_PATH_LENGTH];
    char *src_name = strrchr(src_path, '/');
    src_name = src_name ? src_name + 1 : src_path; // Extract the source file's name
    if (snprintf(dest_path, sizeof(dest_path), "%s/%s", dest_dir, src_name) >= sizeof(d
        send_ack_to_nm(nm_sock, "COPY_FAILED", src_path);
        fprintf(stderr, "Error: Destination path too long.\n");
        return;
    }

    if (!S_ISDIR(src_stat.st_mode)) {
        // Local file-to-file copy
        copy_file(src_path, dest_path);
        send_ack_to_nm(nm_sock, "COPY_SUCCESS", dest_path);
    } else {
        // Handle directory copy if needed
        // ... (implement directory copy logic if necessary)
    }
}

// ... existing code ...
```

nm2

ss2

- P** debug the code so that all elements of the struct `StorageServer` sent by the storage server (ss2) are correctly received by the naming server (nm) and printed.

To debug the code so that all elements of the `StorageServer` struct sent by the storage server (ss2) are correctly received by the naming server (nm) and printed, we need to make the following changes:

1. In the `nm2.c` file, update the `handle_storage_server` function to correctly extract and print all the fields of the `StorageServer` struct:

```
c Copy

void handle_storage_server(int storage_sock, char *buffer) {
    StorageServer new_server;
    char paths[BUFFER_SIZE];

    // Clear out the structures before using them
    memset(&new_server, 0, sizeof(new_server));
    memset(paths, 0, sizeof(paths));

    // Extract IP address, NM port, Client port, and accessible paths
    int sscanf_result = sscanf(buffer, "%s %d %d %[^\\n]",
                                new_server.ip_address,
                                &new_server.nm_port,
                                &new_server.client_port,
                                paths);

    // Ensure to print the information received
    if (sscanf_result < 3) {
        fprintf(stderr, "Error: Invalid registration format. Received: %s\\n", buffer);
        close(storage_sock);
        return;
    }

    // Split paths into an array
    new_server.path_count = 0;
    char *token = strtok(paths, ",");
    while (token != NULL && new_server.path_count < MAX_PATHS) {
        strncpy(new_server.accessible_paths[new_server.path_count], token, MAX_PATH_LENGTH);
        new_server.accessible_paths[new_server.path_count][MAX_PATH_LENGTH - 1] = '\\0';
        new_server.path_count++;
    }
}
```

```
pthread_mutex_lock(&storage_mutex);

if (storage_server_count < MAX_STORAGE_SERVERS) {
    storage_servers[storage_server_count] = new_server;

    printf("Registered Storage Server #%d\n", storage_server_count + 1);
    printf("IP Address: %s\n", new_server.ip_address);
    printf("Naming Port: %d\n", new_server.nm_port);
    printf("Client Port: %d\n", new_server.client_port);
    printf("Accessible Paths:\n");
    for (int i = 0; i < new_server.path_count; i++) {
        printf("    - %s\n", new_server.accessible_paths[i]);
    }
    storage_server_count++;
} else {
    printf("Maximum storage server limit reached.\n");
}

pthread_mutex_unlock(&storage_mutex);
close(storage_sock);
}
```

2. In the `ss2.c` file, update the `register_with_naming_server` function to correctly fill in the `StorageServer` struct fields:

```
c
void register_with_naming_server(const char* naming_server_ip) {
    int sock;
    struct sockaddr_in server_addr;
    StorageServer info;

    // Create a socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Setup server address for connecting to the Naming Server
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port_naming);
    if (inet_pton(AF_INET, naming_server_ip, &server_addr.sin_addr) <= 0) {
        perror("Invalid Naming Server IP address");
        close(sock);
        exit(EXIT_FAILURE);
    }

    // Connect to the Naming Server
    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
```

```

    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr))) {
        perror("Connection to Naming Server failed");
        close(sock);
        exit(EXIT_FAILURE);
    }

    // Get accessible paths from the user
    int n;
    printf("Enter number of accessible paths: ");
    scanf("%d", &n);
    if (n > MAX_PATHS) {
        fprintf(stderr, "Error: Exceeded maximum number of paths.\n");
        close(sock);
        exit(EXIT_FAILURE);
    }
    printf("Enter list of accessible paths:\n");
    for (int i = 0; i < n; i++) {
        scanf("%s", info.accessible_paths[i]);
    }

    // Fill in storage server information
    strncpy(info.ip_address, naming_server_ip, INET_ADDRSTRLEN);
    info.client_port = port_client;
    info.nm_port = port_naming;
    info.path_count = n;

    // Send registration information to the Naming Server
    if (send(sock, &info, sizeof(info), 0) < 0) {
        fprintf(stderr, "Failed to send data to Naming Server.\n");
        close(sock);
        exit(EXIT_FAILURE);
    }
    printf("Registered with Naming Server: IP = %s, Client Port = %d, Naming Port = %d\n",
           info.ip_address, info.client_port, info.nm_port);

    close(sock);
}

```

The main changes are:

1. Updating the `scanf` call in the `handle_storage_server` function to correctly extract all the fields of the `StorageServer` struct.
2. Ensuring the `StorageServer` struct is properly filled in with the correct values in the `register_with_naming_server` function.

With these changes, the naming server should now correctly receive and print all the elements of the `StorageServer` struct sent by the storage server.

ⓘ Claude does not have the ability to run the code it generates yet.

Subscribe to Pro for 5x more usage with Claude 3.5 Sonnet.

Subscribe to Pro ×