Mini-Projet UNIX

Nom : Mohamed Yassine Ahmed Ali & Dhiee Hmam
Classe : 2ING GR 02

Description de l'application

Le programme est une application client-serveur en C, utilisant des tubes FIFO (First In, First Out) pour la communication entre un client et un serveur. Le serveur écoute en permanence les messages du client via les tubes FIFO, répond avec une séquence de nombres aléatoires, et utilise des signaux pour gérer la synchronisation entre les processus.

Structure du Projet

Le projet est divisé en plusieurs fichiers source et répertoires :

- **src** : Contient les fichiers sources principaux (`src_cli_fifo.c`, `Handlers_Serv.c`, `utils.c`, `server.c`, `client.c`).
- **include** : Contient les fichiers d'en-tête pour les déclarations de fonctions.
- **test** : Contient les fichiers pour les tests unitaires, incluant le fichier de test Unity (`unity.c`) et le fichier de tests spécifiques (`tests.c`).
- **build** : Répertoire de construction des fichiers objets et exécutables.

Approche du Serveur

Le serveur exécute les étapes suivantes :

1. Il crée deux tubes FIFO : un pour recevoir des messages du client (`FIFO1`) et un autre pour envoyer des réponses (`FIFO2`).
2. Il attend qu'un message soit envoyé par le client via `FIFO1`.
3. Une fois un message reçu, le serveur génère une séquence de nombres aléatoires et renvoie cette séquence au client via `FIFO2`.
4. Le serveur utilise des signaux pour réveiller le client ou se terminer proprement lorsque le signal approprié est reçu (SIGUSR1 pour réveiller et SIGINT pour terminer).

Approche du Client

Le client effectue les étapes suivantes :

1. Le client génère un nombre aléatoire et l'envoie au serveur via `FIFO1`.
2. Il attend la réponse du serveur via `FIFO2`.
3. Après avoir reçu la réponse, le client envoie un signal SIGUSR1 au serveur pour l'indique qu'il peut continuer ou se terminer.

Utilisation de Signaux

Les signaux sont utilisés pour la synchronisation :

- Le client envoie un signal SIGUSR1 au serveur pour l'indiquer qu'il doit traiter la demande.

- Le serveur envoie un signal SIGUSR1 au client après avoir envoyé la réponse pour l'informer que le traitement est terminé.
- **Génération de nombres aléatoires** : Le programme génère des séquences de nombres aléatoires. Le serveur génère une séquence de nombres en fonction de l'entrée du client.
- **Gestion des erreurs** : Des mécanismes de gestion d'erreurs robustes ont été mis en place pour gérer les échecs de création de FIFO, d'allocation de mémoire et de lecture/écriture dans les fichiers FIFO.

Tests Unitaires Implémentés

Les tests unitaires sont implémentés à l'aide de la bibliothèque **Unity**. Voici les tests réalisés :

1. Test de la création de FIFO

Testé avec la fonction `create_fifo` pour vérifier que le fichier FIFO est bien créé.

```
void test_create_fifo(void) {
    const char* test_fifo = "/tmp/test_fifo";
    create_fifo(test_fifo);
    TEST_ASSERT_EQUAL(0, access(test_fifo, F_OK)); // Vérifie que FIFO
existe
    close_fifo(test_fifo);
}
```

Le test vérifie que le FIFO a été créé correctement en vérifiant son existence.

2. Test d'écriture et de lecture dans un FIFO

Testé avec les fonctions `write_fifo` et `read_fifo` pour vérifier que les messages envoyés par le client sont correctement reçus par le serveur.

```
void* reader_thread(void* arg) {
    message* received_msg = NULL;
    read_fifo(FIFO1, &received_msg);

    TEST_ASSERT_EQUAL(1234, received_msg->pid);
    TEST_ASSERT_EQUAL(5, received_msg->content_size);
    TEST_ASSERT_EQUAL_STRING("test", received_msg->content);

    free(received_msg);

    return NULL;
}

void* writer_thread(void* arg) {
    message* test_msg = malloc(sizeof(message) + 5);
    test_msg->pid = 1234;
    test_msg->content_size = 5;
    strcpy(test_msg->content, "test");
```

```
    write_fifo(FIFO1, test_msg);

    free(test_msg);

    return NULL;
}

void test_write_and_read_fifo(void) {
    create_fifo(FIFO1);

    pthread_create(&reader_thread_id, NULL, reader_thread, NULL);
    pthread_create(&writer_thread_id, NULL, writer_thread, NULL);

    pthread_join(reader_thread_id, NULL);
    pthread_join(writer_thread_id, NULL);

    close_fifo(FIFO1);
}
}
```

Le test vérifie que le message écrit dans le FIFO est bien lu et que les données sont correctement transmises.

3. Test de la génération de nombres aléatoires

Testé avec la fonction `generate_random_number` pour vérifier que le nombre généré est dans l'intervalle attendu.

```
void test_generate_random_number(void) {
    char* random_num = generate_random_number();
    TEST_ASSERT_NOT_NULL(random_num);
    int num = atoi(random_num);
    TEST_ASSERT_TRUE(num >= 0 && num < MAX); // Vérifie que le nombre
généré est dans la plage attendue
    free(random_num);
}
```

Le test vérifie que le nombre aléatoire généré se situe bien dans la plage spécifiée par `MAX`.

4. Test de la génération de séquences de nombres aléatoires

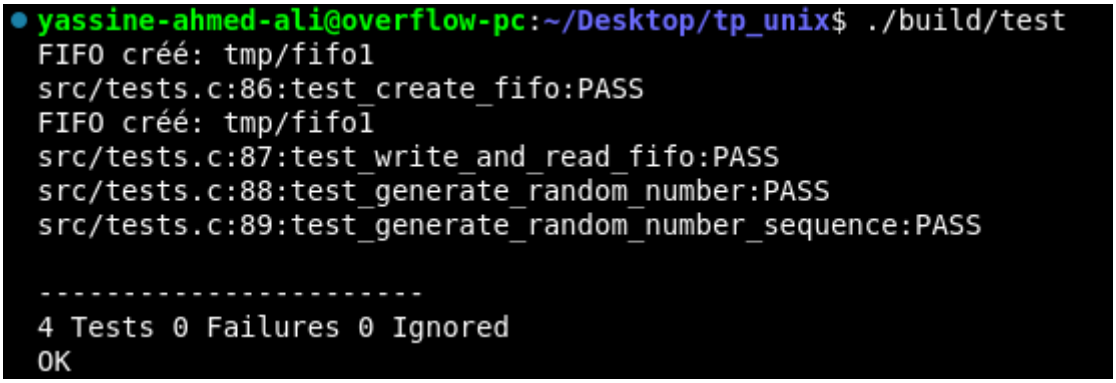
Testé avec la fonction `generate_random_number_sequence` pour vérifier que la séquence générée correspond au nombre spécifié par le client.

```
void test_generate_random_number_sequence(void) {
    char* sequence = generate_random_number_sequence("5");
    TEST_ASSERT_NOT_NULL(sequence);
```

```
    int count = 0;
    for (char* token = strtok(sequence, " "); token != NULL; token =
strtok(NULL, " ")) {
        int num = atoi(token);
        TEST_ASSERT_TRUE(num >= 0 && num < MAX); // Vérifie que chaque
nombre dans la séquence est valide
        count++;
    }
    TEST_ASSERT_EQUAL(5, count); // Vérifie que 5 nombres ont été générés
    free(sequence);
}
```

Le test vérifie que la séquence de nombres aléatoires générée a bien le nombre de valeurs spécifié et que chaque valeur est valide.

Validation des Tests



Conclusion

Le programme met en place une communication client-serveur basée sur des tubes FIFO et des signaux pour la gestion des processus. Des tests unitaires ont été ajoutés pour garantir la stabilité du programme, en testant la création de FIFO, l'écriture/lecture dans les FIFO, et la génération de nombres aléatoires. Les tests sont implémentés à l'aide de la bibliothèque **Unity** et permettent de valider les fonctionnalités critiques de l'application.