Télécommunications

TS226

Rapport

Projet Réseau et Système

Etudiants :
Drystan Groell
Nicolas Blanc

Professeurs :
Guillaume Mercier
Joachim Bruneau-Queyreix

17 Décembre 2021

Table des matières

1 Introduction

Le but du projet était de réaliser un programme partageant de la mémoire entre plusieurs machines de l'ENSEIRB. La première partie du projet consistait à réaliser un lanceur de programme. Son but était de se connecter en ssh aux machines de l'enseirb, de se connecter à elles en utilisant les protocoles TCP et IP, pour ensuite leur envoyer des informations pour qu'ils s'interconnectent. La seconde partie du projet était consacrée à l'interconnexion entre ces différentes machines et à la gestion des erreurs de segmentation.

2 Phase 1

2.1 Lecture du fichier machinefile

La première étape du projet est de lire le fichier machinefile, qui contient le nom des machines de l'école sur lesquelles on éxecute les processus distants. Pour lire le fichier, nous avons fait le choix d'utiliser la fonction **fget** pour lire le fichier caractère par caractère. En effet, les lignes vides dans le fichier ne doivent pas être prises en compte. Ainsi, si deux caractères "**n**" se succèdent, on détermine que la ligne est vide. Ainsi, nous pouvons déterminer le nombre correct de machines textbfnumt_procs sur lesquelles nous devons nous connecter et exécuter les processus distants . Ensuite, pour extraire le nom des machines, nous extrayons chaque ligne du fichier en utilisant **getline**, puis nous retirons la chaîne de caractère "**n**" pour changer le champ **connect_info.machine** du tableau **proc_array** de type dsm_proc_conn_t qui contient toutes les informations sur les processus. Nous avons choisi arbitrairement le rang des machines par rapport à leur position dans le fichier **machinefile**.

2.2 Création des processus distants et redirections des sorties

Une fois le nombre de machines a été déterminé, il s'agit de créer les différents processus et de rediriger leur sorties standard et d'erreur. Nous créons au préalable 2 tableaux de textbfnumt_procs entiers textbffdt_out textbffdt_err. Pour créer les processus, nous utilisons la fonction **fork**. Dans le processus père, nous créons des tubes anonymes dans lesquels les redirections de sorties standard et d'erreur vont être réalisés, puis nous stockons les fd de ces tubes respectivement dans les 2 tableaux. Dans les processus fils, nous fermons les sorties standard et d'erreur afin d'assurer la redirection, puis nous exécutons la commande ssh nous permettant d'exécuter le programme à distance.

2.3 Communication entre les processus distants et dsmexec

Après s'être connecté sur la socket de **dsmexec**, chaque processus distant envoie son nom à **dsmexec** pour qu'il puisse mettre en relation les ports sur lesquels il a accepté les connexions avec le rang du processus distant. Ainsi, la structure **proc_array** stocke toutes les informations de connexion concernant ces processus. Enfin, **dsmexec** envoie à tous les processus le nombre de machines, leur rang ainsi que **proc_array**.

2.4 Difficultés rencontrées

Notre difficulté principale lors de la phase 1 était la redirection des sorties standards et d'erreur. Nous n'arrivions pas à bien recevoir de manière cohérente les informations envoyées par les processus distants.

3 Phase 2

3.1 Phase d'interconnexion des processus distants

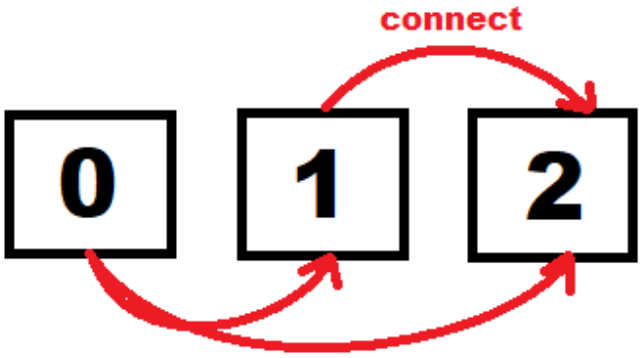

FIGURE 1 – Schéma représentant l'interconnexion entre les processus

Pour réaliser l'interconnexions entre chaque processus, nous réalisons une chaîne. Chaque processus se connecte à tous les processus de rang supérieur, mais ils doivent d'abord accepter la connexion des rangs inférieurs (voir la figure ci-dessus). Lors de l'utilisation de la fonction connect, il est nécessaire d'utiliser une boucle while pour que le processus tente de se connecter tant que la connexion n'a pas été refusé. Ainsi, cela permet d'assurer la connexion entre tous les processus. Cependant, il est impossible pour un processus qui accepte de connaître quel processus s'est connecté à lui. Chaque processus envoie son rang après s'être connecté pour y remédier. Tous les descripteurs de fichiers sont stocké dans le tableau **sockets_array** de taille **DSM_NODE_NUM**.

3.2 Gestion des requête

3.2.1 Requêtes pour récupérer une page

Lorsqu'un processus essaye d'accéder à une page qu'il ne possède pas, une erreur de segmentation se produit. Alors, le traitant du signal va envoyer un requête **DSM_REQ** au processus propriétaire de la page. La gestion du signal **SIGSEGV** se fait alors en 3 étapes :

— 1ère étape, récéption de la requête par le propriétaire **DSM_REQ** : Après avoir vérifié que le processus qui reçoit la requête est bien le propriétaire de la page demandée. Il envoie cette même page à la source de la reqûete avant de libérer l'espace mémoire en utilisant la fonction **dsm_free_page**.

— 2ème étape, récéption de la page par le demandeur **DSM_PAGE** : Le thread du demandeur qui reçoit les reqûetes va devoir alloué l'espace mémoire de la page attendue avec la fonction **dsm_alloc_page**, avant de recevoir la page.Puis il faudra modifier son propriétaire et envoyer à chaque processus une requête de type **DSM_NREQ** qui aura pour but de changer le propriétaire de la page.
— 3ème étape (**DSM_NREQ**) : Cette étape consiste juste à changer le propriétaire de la page dont le numéro est mis dans la reqûete en utilisant la fonction **dsm_change_info**.

3.2.2 Requête pour libérer les ressources alloués pour la DSM

Dès que toutes les pages sont allouées et utilisées correctement, **DSM_FINALIZE** est envoyée à chaque processus, supprimant ainsi chaque connexion réalisée entre les processus. Nous utilisons la variable globale **proc_active**, correspondant au nombre de processus n'ayant pas encore terminé. Lorsque qu'un processus reçoit la requête **DSM_FINALIZE**, il désincrémente **proc_active** une fois. Dès que **proc_active** vaut 0, chaque processus arrête de regarder ses ports dans la fonction **poll** et rentre dans la fonction **dsm_finalize** pour fermer toutes les connexions.

3.3 Synchronisation entre dsm_comm_daemon et dsm_handler

Un outil de synchronisation est nécessaire pour éviter que le traitant de signal envoie une requête de type **DSM_REQ** en permanence. Nous avons fait le choix d'utiliser un sémaphore car c'était l'élément de syncrhonisation le plus facile à implémenter, mais nous aurions pu également utiliser un verrou. Nous initialisons le sémaphore avec aucun jeton à la fin de **dsm_init**. Pour bloquer le traitant de signal juste avant sa terminaison, nous utilisons la fonction **sem_wait**, qui va attendre qu'un jeton soit disponible pour le prendre et continuer sa section de code. Le jeton est donné lors de la récéption de la requête **DSM_PAGE**. Ainsi, la page est correctement alloué et une erreur de segmentation ne se produit pas à nouveau.

3.4 Difficultés rencontrées

Notre difficulté principale durant la phase 2 était le protocole d'interconnexion. En effet, nous n'avions pas compris qu'un processus devait rééssayer de se connecter tant que la connexion n'avait pas réussie. De plus, nous réalisions le double de connexion au départ, c'est à dire que chaque processus se connectait et acceptait **DSM_NODE_NUM** connexions. Notre second problème rencontré a été celui lors de la récéption de la requête **DSM_PAGE**. Nous n'accédions pas correctement à la page, même si la page était correctement réallouée au nouveau processus car nous avions oublié de alloué la page avec mmap à celui qui la recevait et de la libérer pour celui qui l'envoyait avec munmap.

4 Conclusion

Finalement, ce projet nous a permis de progresser dans des notions comme la redirection de descripteurs de fichiers, la connexion via des sockets etc. Même si nous ne sommes pas parvenus

à la fin des 2 phases, nous sommes satisfaits de notre progression car nous n'étions pas loin de les finaliser.