Système et Réseau Implémentation d'une Mémoire Partagée Distribuée (DSM) :

Rapport de Projet

Telecommunication-ENSEIRB-MATMECA

Hafidi Aymane,
Nhairech Oussama

Encadré Par :
Guillaume Mercier , Joachim Bruneau-Queyreix

1 Introduction

Dans le calcul numérique, il s'avère parfois utile d'avoir plusieurs unitées de calcul qui partagent une mémoire virtuelle. Ce système de partage s'appelle la DSM : distributed shared memory. Dans ce projet, une tentaive sera présentée. Ce projet est réparti en deux phases : une partie pour l'unité centrale dsmexec et une pour le dsm.

2 Phase 1 : Lancement des processus

2.1 DSMEXEC :

2.1.1 Création des processus

Depuis un fichier `machine_file` qui contient le nom des machines distantes, Une boucle a été mise pour avoir `num_procs` processus fils. À l'aide de la fonction `execvp("ssh", argv2)`, chacun peut créer à son tour un processus distant en éxcecutant `dsmwrap` avec une liste d'arguments choisies judicieusement. Ces arguments sont le `hostname` de la machine locale, son port , l'éxecutable et bien évidemment ses arguments.
Afin de centraliser les processus, un tableau de structure `proc_array` a été mis en place pour stocker les informations de chaque machine. En fait, cette structure est remplie périodiquement dans le `dsmexec` à l'aide de la socket d'initialisation.

2.1.2 Socket d'initialisation

Cette socket crée entre le processus pére et le processus distant utilise le protocle **TCP** pour assurer la fiabilité. elle a été utiliser dans un premier temps pour compléter le remplissage de `proc_array`. Par la suite le processus pére envoie à chaque processus distant son rang,le numéro de processus et le tableau `proc_array` à travers cette socket.

2.1.3 Centralisation et filtrage des affichages

Il s'agit d'afficher les messages des sorties standard et d'erreur des processus distants. cela nécessite une communisation inter-processus, pour ce faire des tubes anonymes ont été utiliser dès que le problème de synchronisation inter-processus n'existe pas.
Avant la création de chaque processus local par **fork**, deux tubes ont été créer pour que le processus local puisse les hériter. L'étape suivante était la redirection des extrimités des tubes. Pour le processus père il s'agit de fermer les extrimités d'écriture, tandis que pour les processus crée il fallait premièrement fermer les éxtremités en lecture et de rediriger la sortie standard et d'erreur vers l'extrémité en écriture des tubes à l'aide de la fonction `dup2` pour que la redirection se fait de manière **atomique**.

La fonction **poll** a été utilisé pour assurer l'affichage, pour que la lecture ne ce fait que lorsqu'il y a une **activité** sur l'une des extrémités des tubes. Et en connaissant l'extrémité active on peut filtrer le message (Sortie standard ou d'erreur, le processus qui a écrit ce message).

2.2 DSMWRAP

Le rôle de `dsmwrap` est de nettoyer les arguments reçus de la part de `dsmexec`. Le port et le hostname sont utilisés pour créer la socket d'intialisation (*cf. 1.2*)et d'envoyer les informations nécessaires pour le remplissage de `proc_array` à savoir le nom,le pid et le port de la socket créer pour connecter les processus distants ultérieurement.
Après, la fonction `execvp(argv_exec[0], argv_exec);` est utilisé pour éxecuter l'éxecutable finale avec le reste des arguments.

3 Phase 2 : Mise en place de la DSM

3.1 dsm_init

3.1.1 Connexion inter-processus

Avant de connecter les processus entre eux la fonction reçoit les informations envoyer par **dsmexec** (*cf. 1.2*). La démarche utilisée pour connecter les processus entre eux est la suivante : le processus accepte les demandes de connexion parvenant des processus de rang supérieur à le sien et se connecter au processu qui ont un rang inférieure.
Lorsqu'un processus se connecte envoi son rang que le processus reçoit aprés avoir accepter la connexion, ce mécanisme est mis en place pour remplir correctement `proc_array`. Car au moment de l'acceptation, le processus n'a aucune idée sur le processus qui s'est connecté.

3.1.2 Commuunication inter-processus

un processus doit à la fois exécuter son code et communiquer avec les autres processuses. Pour ce fait un thread a été utiliser pour la communication bien évidemment à l'aide de la fonction **poll**. Lorsqu'un processus envoie une requête à un autre ce dernier aura une activité sur une des sockets de connexion qui va étre détectee par la fonction poll, le processus va recevoir la requête et réagir d'aprés son type :

- DSM_REQ : désallouer la page mémoire et envoyer la page (Requête DSM_PAGE) au processus demandeur
- DSM_PAGE : allouer la page mémoire, mettre à jour le propriétaire de la page dans ça table et envoyer un broadcast pour que les autres processus mettent à jour leur table(Requête DSM_NREQ) en changeant le propriétaire de la page.
- DSM_NREQ : mise à jour la table des propriétaire le propriértaire de la page `page_num` et maineant le processuss de rang `source`
- DSM_FINALIZE : signifie que le processus de rang `source` est prêt à ce terminer.

3.2 le traitent de signal SIGSEGV

Quand un processus essaye de faire une opération(Lire, écrire,exécuter) dans une page qui n'appartient pas à lui, une erreur de type `SIGSEGV` est déclenchée. Un traitent de signal est mis en place pour détecter cela, il appelle la fonction `dsm_handler`, cette derniere prend comme argument le numéro de la page qui a induit l'erreur.
Cette erreur est traitée une seule fois à l'aide de la variable globale `DEMENDING_PAGE` qui devient 1 après l'envoi (ligne 240). Ce dernier s'agit d'envoyer la structure `dsm_req_t` au propriétaire de la page de type request `DSM_REQ`. Ainsi le traitement commence entre les deux processus à l'aide de la communication inter-processus et la fonction `void Req_handler(int sock_fd)`. La démarche suit pour que ne l'envoi de la demande ne se fait qu'une seule fois était avec la variable `DEMENDING_PAGE`. Mais cela signifie que l'erreur de segmentation se produit toujours mais le `Dsm_handler` la ignore puisque il a déjà demandé la page.

Autre démarche possible : faire une boucle d'attente sur la variable `DEMENDING_PAGE`, cela signifie que le programme va rester bloquer dans la fonction `Dsm_handler` jusqu'à recevoir la page.

3.3 dsm_finalize

Quand le processus est prêt pour se terminer il notifie tous les autres en envoyant un message de type `DSM_FINALIZE` après qu'il incrémente la valeur partagée et globale `DSM_READY_TO_DISC`. Quand cette valeur atteint le nombre de processus ce qui veut dire que tout le monde est prêt pour se terminer, les threads peuvent se teminer avec la fonction `pthread_join(comm_daemon,NULL)`. Il reste que libérer les mallocs et fermer les descripteurs fichiers proprement.

4 Conclusion

En gros, ce projet fait l'appel à plusieurs notions du cours programmation système notamment les sockets de communication, les tubes, les threads etc. On a pu plus au moins les concrétiser dans ce projet. Cette tentative de DSM ne fonctionne pas parfaitement. Puisque après l'échange de propriété, on n'a pas envoyé les données de la page concernée.