Sujet du Projet Système et Réseau (PR204) Implémentation d'une Mémoire Partagée Distribuée (DSM)

Joachim Bruneau-Queyreix, Guillaume Mercier, Philippe Swartvagher

joachim.bruneau-queyreix@labri.fr,
guillaume.mercier@enseirb-matmeca.fr,
philippe.swartvagher@enseirb-matmeca.fr

2024

1 Objectif et contexte du projet

Le but de ce projet est de mettre en place un logiciel permettant de partager de la mémoire virtuelle (i.e des plages d'adresses) entre plusieurs processus répartis sur différentes machines physiques. Cette plage d'adresses (qui sera identique pour tous les processus) sera divisée en un ensemble de *pages mémoire*, c'est-à-dire un ensemble de blocs de 4ko. Tous les processus pourront donc travailler sur ces pages, c'est-à-dire lire et écrire des données à l'intérieur mais seul un processus sera le *propriétaire* d'une page à un instant donné : le processus propriétaire possède la page dans son espace d'adressage et y accède normalement, tandis que si les autres processus essayent de lire ou d'écrire à l'intérieur, cela provoque une erreur de segmentation (i.e `segfault`, signal numéro `SIGSEGV`, 11).
Quand cela arrive, il va s'agir de rattraper ce signal à l'aide d'un *traitant* adapté qui va :

1. déterminer l'adresse qui a provoqué la délivrance du signal ;
2. en déduire le numéro de page concernée ;
3. en déduire le processus actuellement propriétaire ;
4. envoyer une requête à ce processus pour qu'il envoie la page au processus demandeur, qui devient donc le *nouveau propriétaire* de la page mémoire concernée.

Bien entendu le propriétaire actuel devra mettre à jour ses informations concernant cette page et libérer cette dernière de son espace d'adressage tandis que le processus demandeur doit recevoir la page, l'allouer et mettre à jour ses informations concernant cette dernière. Ces informations devront être également être répercutées aux autres processus bien évidemment, afin que tous possèdent des informations cohérentes sur l'état du système. Enfin, le processus demandeur va reprendre son exécution, mais comme il possède désormais la page mémoire (elle fait maintenant partie physiquement de son espace d'adressage) il peut travailler dessus.
Voyons maintenant comment vous allez mettre en place ce logiciel.

2 Hypothèses sur le système à réaliser

Vous allez supposer les choses suivantes pour implanter votre système de *Distributed Shared Memory* (ou DSM) :

— tous les processus ont accès au même système de fichier ;

— le système est limité à `PAGE_NUMBER` pages de mémoire ;
— la taille d'une page est de `PAGE_SIZE` octets (cette taille pourra éventuellement être définie avec `sysconf(_SC_PAGE_SIZE)`) ;
— un processus peut connaître son *numéro de rang* grâce à une variable `DSM_NODE_ID` ;
— un processus peut connaître le nombre total de processus grâce à une variable `DSM_NODE_NUM` ;
— les *rangs* des processus sont numérotés de façon contiguë de 0 à `DSM_NODE_NUM` -1 ;
— tous les processus sont connectés les uns avec les autres. Chaque processus est donc capable de dialoguer directement avec n'importe quel autre processus par le biais de son numéro de rang ;
— l'allocation des pages est faite cycliquement (cf figure 1), et la plage des adresses dans laquelle vous travaillerez sera comprise entre `BASE_ADDR` et `TOP_ADDR`. **Cette plage d'adresses est identique pour l'ensemble des** `DSM_NODE_NUM` **processus** ;
— l'accès à une page mémoire se fait par l'intermédiaire d'un numéro (qui va de 0 à `PAGE_NUMBER` -1) ;
— chaque processus dispose d'une table où sont stockées des informations sur les pages mémoire : propriétaire de la page, état de la page le cas échéant (ce pourra être un tableau `table_pages` par exemple).

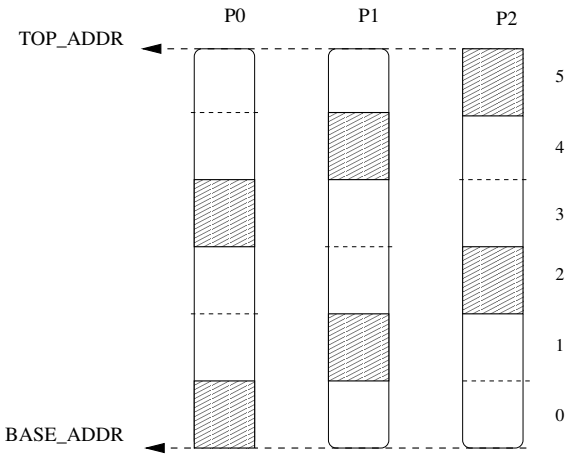

FIGURE 1 – Répartition des pages

La figure 1 montre un exemple d'allocation de départ. Sur cet exemple, le processus 0 possède les pages 0 et 3, le processus 1 possède les pages 1 et 4 et le processus 2 possède les pages 2 et 5.

3 Phase 1 : Lancement des processus

Ce projet va comporter deux phases, qui sont (plus ou moins) indépendantes l'une de l'autre. Un matériel de départ vous sera fourni pour chacune de ces deux phases.

3.1 Syntaxe et la ligne de commande, arguments

Avant de commencer le travail sur ce système de DSM proprement dit, il vous faut développer un programme important, dont le rôle sera de lancer les processus de la DSM sur les différentes machines. Ce programme va prendre en arguments :

1. un fichier contenant les noms des différentes machines sur lesquelles vous allez lancer les processus. On considérera que le nombre d'entrées dans le fichier de machines sera égal au nombre de processus à lancer. Un nom de machine peut apparaître plusieurs fois dans ce fichier.

2. un nom de programme exécutable (ainsi que ses éventuels arguments)

Pour fixer les idées, imaginons que l'on souhaite lancer sur trois machines (toto, tata et titi) un programme appelé `truc` qui utilise la DSM et qui prend en argument `arg1`, `arg2` et `arg3`. Dans ce cas, notre commande sera la suivante :

```
dsmexec machinefile truc arg1 arg2 arg3
```

Avec le fichier `machinefile` qui contient les lignes :

```
> cat machinefile
toto
tata
titi
```

À la suite de l'exécution de cette commande, le processus 0 s'exécute sur la machine `toto`, le processus 1 sur `tata` et le processus 2 sur `titi`. Bien évidemment, cela suppose que ces machines partagent un même système de fichier (ex : NFS) et que le programme `truc` est *accessible* (i.e visible) sur toutes les machines considérées.

N.B : Le programme `dsmexec` peut tout à fait s'exécuter sur une machine qui n'est pas listée dans le fichier `machinefile` pris en argument.

N.B (2) : On fera attention à gérer convenablement les lignes vides dans le fichier `machinefile` (c'est-à-dire ne pas les prendre en compte ...).

3.2 Fonctions et capacités du lanceur `dsmexec`

Outre son rôle de lanceur de processus, `dsmexec` aura plusieurs autres fonctions et capacités importantes :

- il sera notamment chargé d'affecter les numéros (rangs) aux différents processus utilisant la DSM ;
- il sera chargé d'envoyer les données nécessaires pour les connexions à tous les processus distants via des sockets ;
- il devra *centraliser et filtrer* les affichages des *sorties standard et d'erreur* de tous les processus distants. Par exemple, en supposant que l'on exécute le programme `truc` qui affiche "Hello World !" sur sa sortie standard, voici un exemple de filtre :

```
[Proc 0 : toto : stdout] Hello World !
[Proc 1 : tata : stdout] Hello World !
[Proc 2 : titi : stdout] Hello World !
```

Avec le programme `truc` suivant :

```
#include "stdio.h"

int main(int argc, char *argv[])
{
  fprintf(stdout,"Hello World!\n");
  return 0;
}
```

3.3 Architecture du lanceur `dsmexec`

Le processus `dsmexec` va donc devoir créer un ensemble de processus *locaux* (ie situés sur la même machine que lui) avec lesquels il communiquera via une paire de *tubes* : un tube pour récupérer les informations de sortie standard et un autre pour récupérer les informations de sortie standard d'erreur. Ces processus vont devoir exécuter une commande `ssh` afin de créer les processus *distants*. En ce qui concerne la redirection de `stdout` et `stderr` des processus distants vers les processus locaux, elle se fait automatiquement avec `ssh`. Vous allez donc devoir gérer uniquemeent les redirections des processus locaux exécutant

les commandes `ssh` vers le processus `dsmexec`. La figure 2 vous montre un exemple avec trois processus distants lancés sur les machines `toto`,`tata` et `titi`.

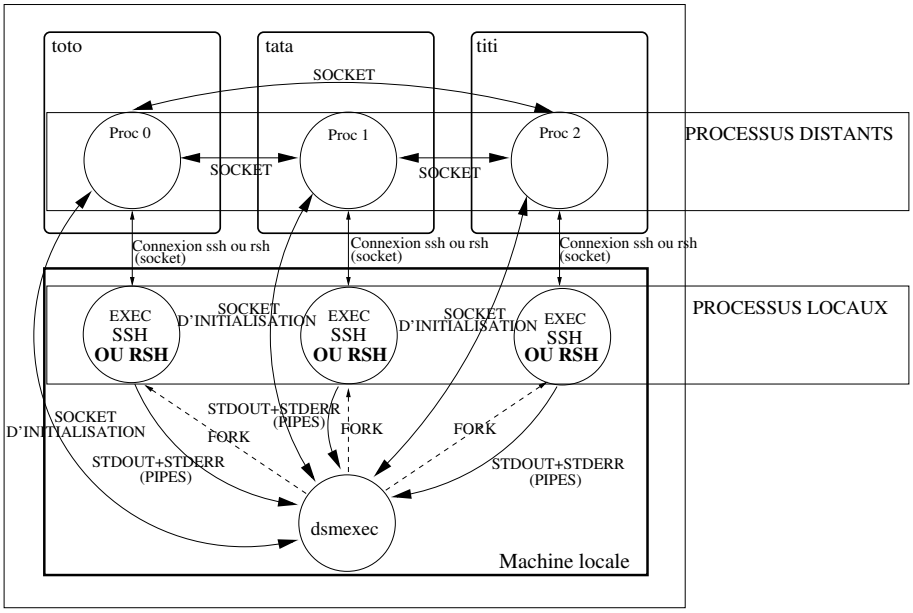

FIGURE 2 – Architecture

Les processus distants (créés avec `ssh`) ne vont pas exécuter immédiatement la commande passée en argument à `dsmexec`, mais vont exécuter un programme intermédiaire, appelé `dsmwrap`, dont le rôle consiste à "nettoyer" la ligne de commande avant d'exécuter la bonne commande. C'est à dire que certains arguments passés à `ssh` vont être utilisés par `dsmwrap`, alors que la commande finale n'en n'a pas besoin. L'utilisation d'un programme intermédiaire va permettre de créer un programme `dsmexec` qui peut lancer tout type d'exécutables et pas seulement des programmes de DSM (qui font appel à `dsm_init` donc).
De plus, Le processus `dsmexec` va communiquer pendant la phase d'initialisation avec tous les processus distants via des sockets temporaires que l'on pourra fermer une fois cette phase terminée. Ensuite, ce processus va passer son temps à attendre que des données (des *caractères*) arrivent sur les tubes dédiés aux événements sur `stdout` et `stderr`.

3.4 Matériel de départ

Le matériel fourni pour cette phase est le suivant :
— un fichier `dsmexec.c` qui contient le squelette du lanceur ;
— un fichier `dsmwrap.c` qui contient le squelette du processus intermédiaire qui va "nettoyer" la ligne de commande avant de lancer le programme final sur chaque machine
— un fichier `common_impl.h` qui contient certains prototypes de fonctions utiles pour le développement des programmes `dsmexec` et `dsmwrap` ainsi que certaines structures de données.
— un fichier `common.c` qui contient l'implémentation de fonctions utiles, utilisées à la fois dans `dsmexec` et `dsmwrap`
— Un Makefile
— Un programme d'exemple
Un répertoire `bin` est créé automatiquement lors de la compilation dans lequel tous les exécutables sont copiés. Ce répertoire est recréé si jamais il est effacé (par ex. avec la commande `make veryclean`).

3.5 Interopérabilité entre les différents lanceurs `dsmexec`

Afin de garantir n'importe quel programme `dsmexec` développé par un binôme sera être en capacité de communiquer avec les programmes de DSM développés par n'importe quel autre binôme lors de la phase 2, certaines structures de donnéess sont fixées et **ne peuvent pas être modifiées**. De même, le code du `dsmexec` est structuré à certains endroits d'une façon qu'il n'est pas possible de modifier.

Enfin, pour vous faciliter la vie avec les problèmes d'emplacement de programmes exécutables sur les machines distantes, vous allez définir une variable d'environnement appelée `DSM_BIN` (le nom est imposé) qui contiendra le nom du chemin vers le répertoire `bin` de votre projet où sont copiés tous les exécutables compilés. Ce chemin sera également copié dans votre variable d'environnement `PATH` afin que le système trouve tout seul le bon chemin vers les exécutables stockés dans `DSM_BIN`.

Pour ce faire, il faut éditer votre fichier `login/.bashrc` et y mettre les définitions suivantes :

```
export DSM_BIN=/mon/chemin/vers/mon/projet/bin
export PATH=$DSM_BIN:$PATH
```

N'oubliez pas de sourcer votre fichier `.bashrc` après l'avoir édité !

Vous pouvez tester ensuite si la manœuvre a fonctionné en tapant les commandes :

```
echo $DSM_BIN
```

ou

```
which dsmexec
```

afin de vérifier la définition des chemins.

N.B : n'oubliez pas que vous pouvez utiliser les fonctions `execvp`, `execvpe` et `execlp` dans votre projet ...

4 Phase 2 : Mise en place de la DSM

Une fois votre lanceur `dsmexec` opérationnel, vous allez mettre en place la bibliothèque de DSM. Dans cette partie, une ébauche de code vous sera fournie pour commencer à travailler.

4.1 Que reste-il à faire ?

Les processus vont utiliser une bibliothèque que vous allez créer plus tard. En fait, cette bibliothèque sera assez réduite puisqu'elle ne va contenir *a priori* que deux fonctions :

1. `char *dsm_init(int argc, char **argv).`
   La fonction `dsm_init` renvoie un pointeur : c'est le pointeur marquant le début de la zone de mémoire distribuée.
2. `void dsm_finalize( void ).`
   La fonction `dsm_finalize` permet de libérer les ressources éventuellement allouées par la DSM.

Vous avez déjà commencé à travailler sur la fonction `dsm_init` puisque les processus doivent être connectés les uns avec les autres. Le code de cette fonction doit maintenant être complété pour effectuer l'allocation par tourniquet des pages du système. La suite du travail va consister à :

— mettre en place le bon traitant de signal pour rattraper les erreurs de segmentation (conseil : lisez bien le manuel pour la fonction `sigaction`)

— mettre en place les envois/réceptions de requêtes.En ce qui concerne les requêtes, nous pouvons en identifier au moins trois types :

1. les messages de requête pour une page manquante ;

2. les messages pour envoyer une page à un processus demandeur ;
3. les messages concernant la mise à jour des informations de la table des pages.

— garantir que les informations concernant les pages (état, propriétaire,etc.) sont bien bien cohérentes d'un processus à l'autre.

4.2 Communications inter-processus

Vous allez travailler avec un ensemble de processus localisés sur des machines différentes et qui vont devoir communiquer entre-eux. Ces communications (i.e demandes de requêtes, réponses, etc.) se feront par le biais de *sockets UNIX*. Vous pouvez utiliser le protocole qui vous paraît le plus adapté pour réaliser le projet (TCP ou UDP).
De plus, Un processus doit à la fois faire ses calculs, et répondre aux requêtes émanant des autres processus. En ce qui concerne la réception de requête, il peut être utile d'envisager la mise en place d'un thread dédié "écoutant" les requêtes pour chaque processus et les traitant quand il les reçoit. Faites-donc attention car vos processus vont donc être très probablement multithreadés (n'oubliez pas `-DREENTRANT` !).

4.3 Matériel de départ

Le matériel fourni est composé des fichiers suivants :

— un fichier `dsm.h`, qui contient les prototypes des fonctions de DSM utilisées par les programmes d'exemple. C'est dans ce fichier que se trouve l'*interface* de la bibliothèque de DSM.
— un fichier `dsm.c` qui contient l'implémentation des fonctions de DSM qui peuvent être utilisées par les programmes d'exemple. Si vous avez besoin, vous pourrez néanmoins déclarer des fonctions non directement utilisables par les programmes d'exemple dans ce fichier.
— Un Makefile et un programme d'exemple

Vous allez donc travailler essentiellement sur les deux fichiers `dsm.c|h` pour finir le projet. Cependant, il vous sera peut-être nécessaire de revenir de temps à autre sur le code du lanceur `dsmexec` si besoin est.

5 Un petit exemple

Comment tout cela fonctionne-t-il ? Examinez le programme suivant (on suppose qu'il s'exécute sur deux machines différentes) :

```
#include ``dsm.h''

int main(int argc, char **argv)
{
  char *pointer;
  char *current;
  int value;

  pointer = dsm_init(argc,argv);
  current = pointer;

  printf(``[%i] Coucou, mon adresse de base est : %p\n'',DSM_NODE_ID, pointer);

 if (DSM_NODE_ID == 0)
  {
   current += 4*sizeof(int);
   value = *((int *)current);
   printf(``[%i] valeur de l'entier : %i\n'',DSM_NODE_ID, value);
```

```
  }
 else if (DSM_NODE_ID == 1)
  {
   current += 16*sizeof(int);
   value = *((int *)current);
   printf(``[%i] valeur de l'entier : %i\n'',DSM_NODE_ID, value);
  }
  dsm_finalize();
  return 1;
}
```

La fonction `dsm_init` renvoie un pointeur : c'est le pointeur marquant le début de la zone de mémoire distribuée. Vous allez donc manipuler des variables situées dans cette zone.

6 Pour aller plus loin : Projet++

Vous avez terminé, vous vous ennuyez ? Essayer de répondre aux problèmes suivants :

— remplacer les sockets par autre chose dans le cas de processus distants situés sur une même machine : segments de mémoire partagée, tubes, etc ...

— modifier la version courante pour intégrer la différence entre accès en lecture et accès en écriture sur les pages : mise en place de plusieurs protocoles

— faire une partie de monitoring des applications (nb de misses) et mesurer les effets de différentes allocations de base (RR, aléatoire, etc.)

— mettre en place des connexions dynamiques (à la demande) au lieu de connecter tous les processus en début de session.

— ❄ —