ENSEIRB-MATMECA
Télécommunications Semestre 7

PR204 - Projet Réseaux & Système

Réseaux & Système

Fait par :

MINIER Pierre

Encadrant :

BRUNO-QUEREIX Joachim
MERCIER Guillaume

Décembre 2021

Sommaire

1 Lancement des processus

1.1 Récupération des affichages des processus distants

1.1.1 Lecture du machine_file

Le fichier machine_file contient le nom des hôtes qui recevront un processus DSM. Nous lisons ce fichier deux fois :

— une première fois pour dénombrer le nombre de lignes non vides, pour initialiser un tableau de structure de type *dsm_proc_t*.

— une seconde fois pour commencer à remplir cette structure avec les noms des machines.

1.1.2 Redirection des sorties standards

On dispose de descripteurs de fichiers stockés dans les tableaux *fd_pipe_stdout* et *fd_pipe_stderr*. Les $2 * i^{èmes}$ et $2 * i^{èmes} + 1$ éléments correspondent à l'entrée et à la sortie du tube liant le futur processus DSM de rang i au lanceur. Pour cela, on redirige les flux de sorties et d'erreurs sur les descripteurs de fichiers associés aux entrées en écriture de tube.

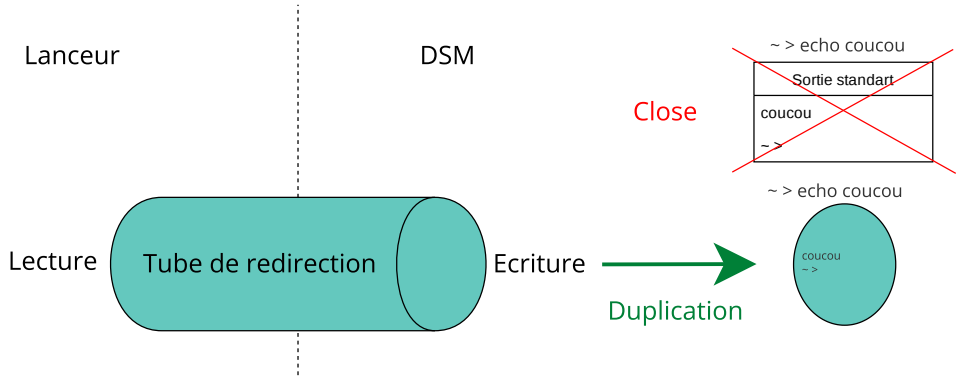

FIGURE 1.1 – Principe de la redirection

Ainsi, le lanceur n'aura plus qu'à lire sur le $i^{ème}$ tube associé à la sortie standard pour récupérer ce qu'écrit le $i^{ème}$ processus sur cette même sortie.

1.1.3 Lecture des tubes de redirection

La lecture des tubes ne se fait pas avec un *poll* sur l'ensemble des descripteurs de fichiers mais grâce à des threads. Un thread est assigné à la sortie en lecture de chaque tube précédemment créé (ce qui fait 2N threads pour N processus). Tous exécutent la même routine : *print_routine*. Celle-ci enregistre les caractères lus sur son tube dans un buffer. Le contenu du buffer est affiché sur la bonne sortie lorsqu'un "\n" est lu ou lorsque le buffer est rempli. Lors de chaque affichage, la routine prend un verrou commun à tous les threads afin d'éviter des mélanges.

1.2 Protocole d'échange

1.2.1 Connexion lanceur - fils

Le programme dsmwrap est un intermédiaire. Lancé sur les machines cibles, il permet de récupérer les informations nécessaires aux futurs communications entre les futurs processus DSM. Il y a donc une connexion a établir entre un dsmwrap et le lanceur.
Côté lanceur, la connexion est établie au moment de la création des processus fils : on crée un fils (qui lance dsmwrap via un ssh) et on accepte sa demande de connexion. Comme la fonction *connect* est bloquante, le lanceur ne pourra pas créer un autre processus avant d'avoir établi la connexion. Cela nous garantit d'associer les rangs des processus distants avec les bonnes sockets de communication. Ainsi, on est certain de ne pas mélanger les informations entre les différents processus.

1.2.2 Échange de données

Une fois toutes les connexions établies, le lanceur vient lire les informations récoltées par les dsmwrap.
Pendant ce temps, les dsmwrap "nettoient" les arguments qui leur ont été passés pour ne garder que les arguments entrés par l'utilisateur. C'est avec ces derniers qu'ils lancent les DSM.
Après avoir centralisé les informations, le lanceur les envoie aux processus DSM. Pour cela il utilise les mêmes sockets. Cela fonctionne car le recouvrement effectué avec *execvp* à la fin de dsmwrap conserve l'état des descripteurs de fichier.

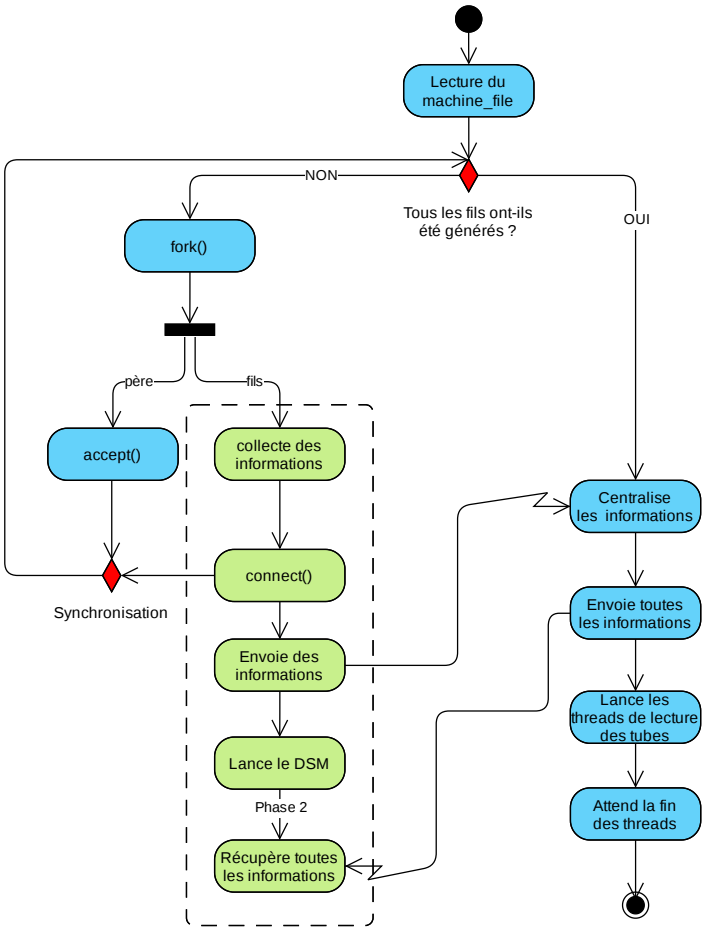

FIGURE 1.2 – Protocole d'échange

2 Mise en place de la DSM

2.1 Initialisation

2.1.1 Récupération des informations utiles

Les variables d'environnement

Un processus DSM donné possède deux sockets :

- dsmexec_fd : pour recevoir les données du lanceur
- master_fd : pour communiquer avec ses frères

Elles sont enregistrées dans l'environnement dans lequel le processus s'exécute. On les récupère donc grâce à la méthode *getenv*.

Les données du lanceur

Sur la socket dsmexec_fd, chaque processus DSM récupère :

- DSM_NODE_NUM : le nombre total de processus DSM
- dsm_bro : un tableau de DSM_NODE_NUM éléments de type dsm_proc_conn_t contenant les informations sur tous les processus frères.
- DSM_NODE_ID : son rang

Toutes ces données sont stockées globalement afin de ne pas surcharger le prototype des fonctions développées par la suite.

2.1.2 Interconnexion des processus frères

La stratégie mise en oeuvre pour interconnecter tous les processus DSM entre-eux est la suivante. Chaque processus DSM accepte les connexions de ses frères ayant un rang inférieur et demande une connexion à ceux ayant un rang supérieur.

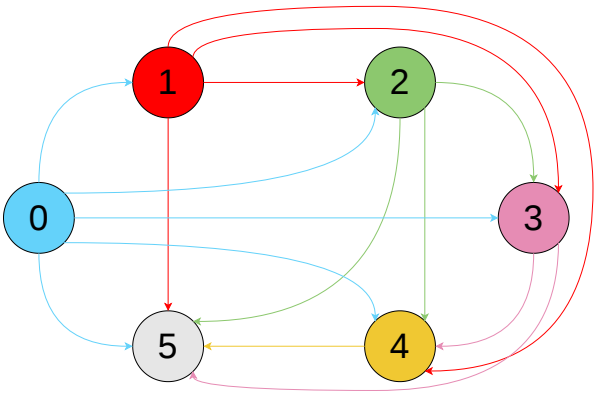

FIGURE 2.1 – Exemple avec 6 processus

Sur la figure 2.1, 0 commence par demander une connexion à 1 et 1 est bloqué jusqu'à ce qu'il reçoive une demande de connexion.
Puis 0 et 1 envoient une demande à 2, qui est bloqué jusqu'à recevoir deux demandes.
Puis 0, 1 et 2 envoient une demande à 3, qui est bloqué jusqu'à recevoir trois demandes, et ainsi de suite...

Chaque *connect* et *accept* génèrent des sockets que l'on stocke dans le tableau dsm_bro à la position du rang du processus avec lequel on a établit une connexion (plus précisément, dans le champ *fd* de la structure associée au processus).

2.2 Manipulation des pages mémoires

2.2.1 Allocation en tourniquet

Les pages utilisées ont leur adresse de mémoire contiguës et sont au nombre de PAGE_NUMBER. Cette borne n'est pas nécessairement égale au nombre de processus : on peut avoir plus de page (voir moins) que de processus. L'attribution se fait à la manière d'un tourniquet : il y a un roulement sur les processus et on leurs associent une page jusqu'à ce que toutes aient un propriétaire.

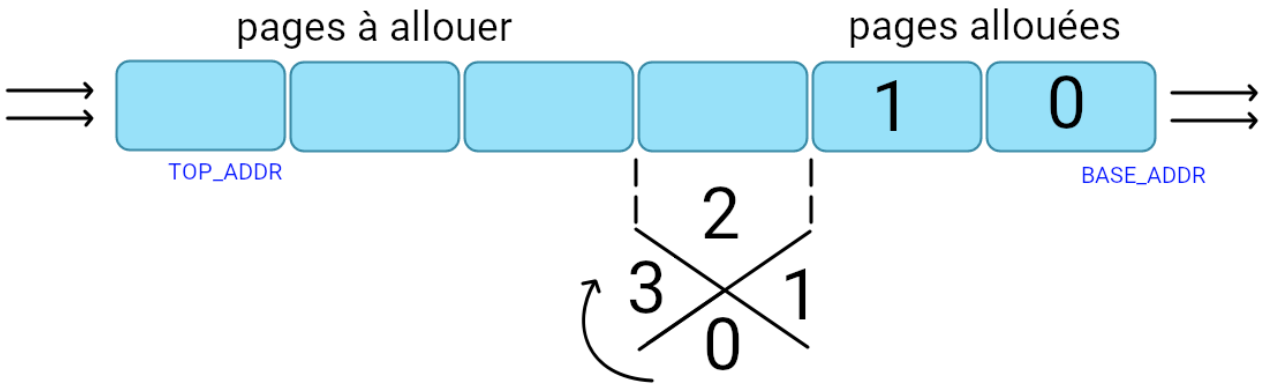

FIGURE 2.2 – Allocation en tourniquet

2.2.2 Les méthodes fournies

Les pages sont caractérisées par leur état (*status*) et leur propriétaire (*owner*). Ces informations sont rassemblées dans le tableau *table_page* dont les indices correspondent aux numéros des pages. Pour manipuler ces pages, on utilise les méthodes statiques suivantes :

- *num2address* : convertit un numéro de page en adresse mémoire
- *address2num* : fonction réciproque à *num2address*
- *address2pgaddr* : indique le début de la page à laquelle se trouve une adresse
- *dsm_change_info* : met à jour *table_page*
- *get_owner* : renvoie le rang du processus propriétaire
- *get_status* : renvoie les droits d'accès sur la page
- *dsm_alloc_page* : alloue une page
- *dsm_protect_page* : paramètre les droits sur une page allouée (aucun / lecture / écriture / exécution)
- *dsm_free_page* : libère une page

2.2.3 Échanger une page

Donner une page

Pour donner une page, on commence par écrire le contenu de la page sur la socket du destinataire. Puis on libère la page de notre mémoire.

Acquérir une page

Pour recevoir une page, on alloue la mémoire correspondante. Puis, on lit le contenu de la page sur la socket de l'émetteur. Enfin, on restreint nos droits à ceux nécessaires.
Bien évidement, cela n'est pas suffisant pour une application en réseau où les échanges de page sont courants. Il faut donc développer un protocole pour s'assurer la bonne réception et la mise à jour continuelle des informations. Cela est l'objet de la prochaine section.

2.3 Traitement d'une erreur de segmentation

2.3.1 Thread principal et daemon de communication

Un processus DSM peut être amené à lire ou à écrire sur une page qui ne lui appartient pas : il reçoit alors une erreur de segmentation et se termine. Le but est plutôt de capturer ce signal et d'attribuer au DSM la page manquante pour qu'il puisse aller au bout de son opération.
Pour cela nous sommes amener à avoir deux flots d'exécution au sein d'un même processus DSM :

- un thread principal : acquiert les informations, alloue ses pages, consulte les pages et termine le DSM.
- un daemon de communication : gère les demandes de pages provenant des autres processus DSM.

En cas d'erreur de segmentation sur la plage mémoire des pages allouées, le thread principal du processus en cause désactive son daemon et communique avec le daemon du processus possédant la page. C'est ce qu'illustre la figure 2.3.

2.3.2 Protocoles

Récupération de l'erreur de segmentation

Les erreurs de segmentations sont toutes étudiées par le traitant *segv_handler* qui récupère les informations suivantes :

- l'adresse mémoire à laquelle l'erreur s'est produite
- le type de tentative d'accès (en lecture ou en écriture)

Si l'adresse mémoire en cause fait partie de la plage sur laquelle sont définies les pages, alors le traitant appelle *dsm_handler*. C'est dans cette fonction que la communication se déroulera.
Sinon, un message d'erreur apparaît et le processus entre dans la phase *finalize* : il attend que les autres DSM se finissent et libère sa mémoire (cf la section 2.4).

Lecture concurrente des messages entre le traitant et le daemon

Lorsqu'il y a une erreur de segmentation, le thread principal va se mettre à envoyer et à recevoir des messages à la place du daemon. Pour que cela se passe sans encombre, le traitant *dsm_handler* commence par prendre un verrou avant d'envoyer un seul message. De l'autre côte, le daemon prend le verrou juste après que la fonction *poll* ait détecté une activité.
Comme la procédure illustrée par la figure 2.3 ne commence qu'après l'envoie du premier message du traitant de signal, on est certain que le daemon ne va pas parasiter la communication.
Côté thread principal, le verrou est débloqué à la fin du traitant ; et du côté du daemon, juste avant d'entrer dans la fonction *poll* (qui est bloquante).

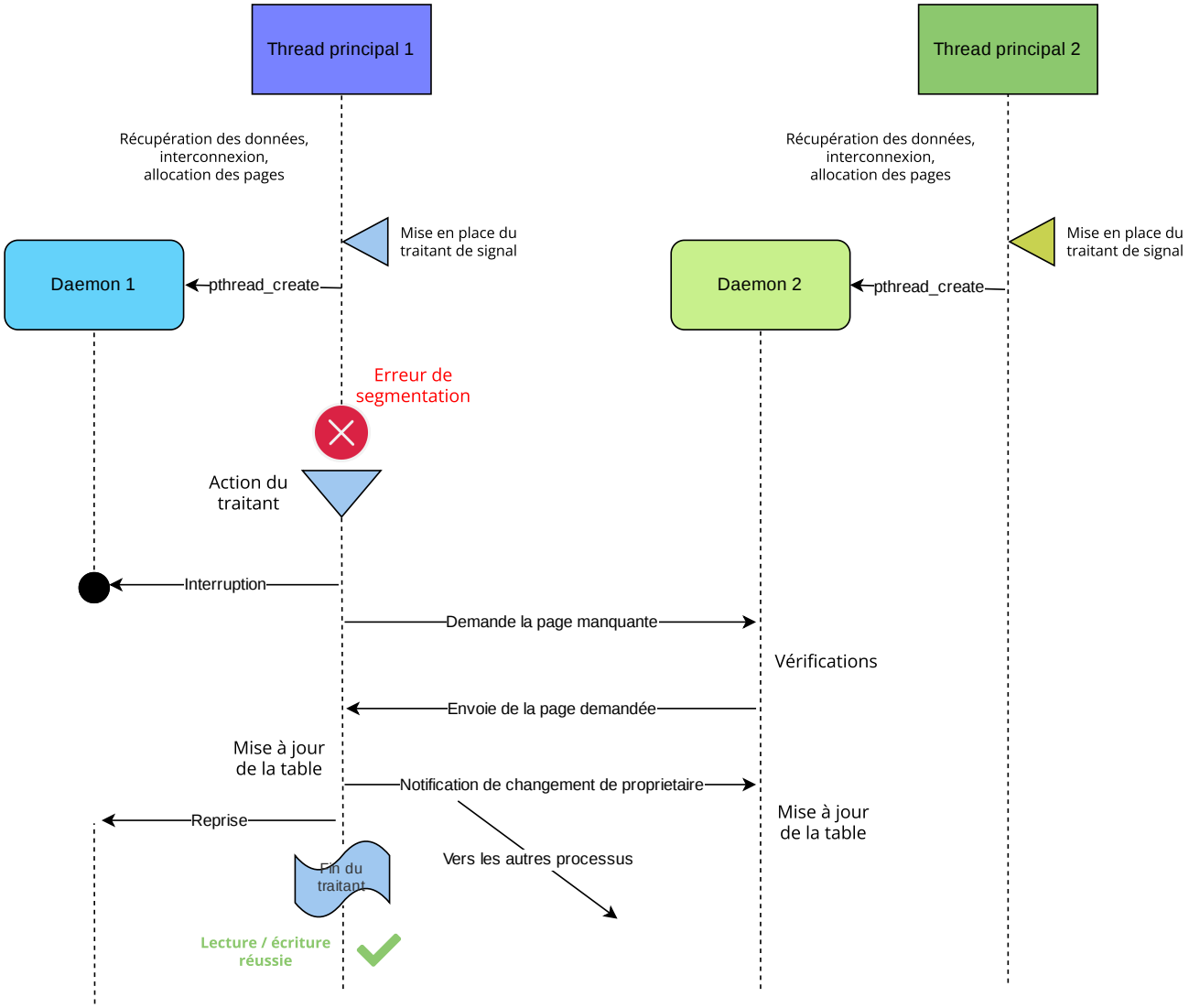

FIGURE 2.3 – Stratégie mise en oeuvre

Demande au propriétaire de la page

Les communications se basent essentiellement sur des messages dont la structure est du type *dsm_req_t*. Le champ *type* permet d'identifier la nature de la requête.
Pour demander une page à un autre DSM, on utilise le type *DSM_REQ*. La daemon du propriétaire va alors vérifier qu'il possède bien cette page.

- Si c'est le cas, le daemon renvoie une requête de conformation (type : *DSM_PAGE*), puis la page comme cela est décrit au point 2.2.3.
- Si ce n'est pas le cas, le daemon renvoie une requête d'erreur (type : *DSM_ERROR*) avec les informations qu'il possède sur le proprietaire de la page demandée. Le traitant corrige alors ses informations et tente un nouveau message.

Notification de changement de propriétaire

Une fois que le traitant de signal a reçu la page (cf point 2.2.3), il met à jour sa table et envoie une requête de type *DSM_CHANGE_PAGE* à tous les autres DSM pour qu'ils actualisent leur table. Enfin, il libère le verrou et le thread principal retente l'action qui a causé l'erreur de segmentation.

2.4 Terminaison des processus

2.4.1 Compte à rebours

Lorsqu'un thread principal arrive à la fin des manipulations sur les pages mémoires qu'il doit effectuer, il envoie à tous les autres processus une requête de type *DSM_FINALIZE*, et attend la terminaison de son daemon grâce à la méthode *pthread_join*.
A la réception de cette requête, les processus daemon décrémentent un compteur indiquant le nombre de thread principal encore en activité. Lorsque ce compteur passe 0, alors ils se terminent et leurs threads principaux respectifs sont débloqués.

2.4.2 fermeture des sockets et libération de la mémoire

Une fois le daemon terminé, toutes les sockets sont fermées : celles permettant de communiquer des processus dsm et master_fd (dsmexec_fd est fermée plus tôt, après la réception des données du lanceur)
Puis on libère les pages que l'on possède, le tableau de données envoyé par le lanceur et on détruit le verrou. libération des pages

2.5 Affichage terminal

Pour faciliter la lecture sur le terminal, les daemons précisent leur nature avec l'entête [Daemon N] où N est le rang du processus et les threads principaux se distinguent avec [N].
De plus, des couleurs sont utilisées afin de remarquer les messages de même nature et avoir une lecture plus efficace. Pour réaliser ces affichages, on utilise la méthode *my_fprintf* qui reprend le prototype de *fprintf* en y ajoutant deux entiers : une couleur et un effet sur le texte. Comme cette fonction a un nombre variable d'argument, on utilise des macros définis dans *stdarg.h*.