DSM logicielle

Des bribes de code sont disponibles pour vous aider dans :
http://runtime.bordeaux.inria.fr/goglin/teaching/SPD/TP-DSM/

1 Défauts de page en espace utilisateur

1. Créer un tampon de 8 pages (bien alignées, cf le man de valloc ou memalign) et utiliser la fonction mprotect() pour qu'une écriture y provoque une erreur de segmentation.
2. Définir un traitant de signal SEGV avec sigaction() et y afficher un message. Vérifier que le programme boucle et comprendre pourquoi.
3. Lors du défaut de page (dans le traitant de signal), restaurer la protection mémoire normale des pages du tampon. Vérifier que cela fait marcher l'écriture comme prévu. Reproduire ce défaut de pages un grand nombre de fois et vérifier le nombre de fautes mineures avec /usr/bin/time.
4. Ajouter le nécessaire pour afficher si le défaut de page corrigé était un accès en lecture ou écriture ainsi que son adresse cible. Autoriser la lecture lors d'un accès en lecture, et autoriser lecture-écriture lors d'un accès en écriture. Modifier le programme pour faire 5 lectures puis 3 écritures puis 2 lectures et vérifier avec des messages de debug que tout se passe bien.
5. Créer un tableau d'attributs pour les pages du tampon en les marquant initialement toutes comme invalides. Modifier la protection d'uniquement la page cible lors d'un accès. Etendre le programme pour réaliser différents accès aux différentes pages et afficher des messages de debug et le contenu du tableau d'attributs après chaque défaut de page.

2 Migration de pages entre processus

1. Une fois le tampon protégé et le tableau d'attributs prêts, créer deux tubes (un IN et un OUT intercroisés) et forker le processus. Bloquer le père en lisant un caractère dans son IN. Le fils écrira ce caractère quand il sera prêt.
2. Dans le père, marquer toutes les pages comme étant accessibles en lecture-écriture. Dans le fils les marquer en invalide. Qu'est-ce que cela signifie sur l'état global de ces pages pour la DSM ? Notez qu'on n'a pas alloué ces tampons mémoire avec des mappings partagés !
3. On ajoute maintenant un protocole pour rapatrier une page du père au fils :
   a) Après son initialisation, le père se bloque en attendant de lire un caractère sur son IN.
   b) Le traitant de signal envoi du numéro de la page (en char) dans OUT puis lit 4096 caractères dans IN.
   c) Lorsque le père reçoit le caractère dans IN, il marque la page correspondante comme invalide puis envoie son contenu dans OUT.
   d) Lorsque le traitant (du fils) reçoit ces 4096 caractères, il les place dans la page correspondante puis la marque en lecture-écriturepuis reprend l'exécution de son accès mémoire.
4. On veut maintenant qu'un thread se charge de traiter les demandes arrivant sur le réseau. Créer un thread dans le père après le fork et le charger d'attendre sur IN et de répondre aux demandes de l'autre processus sur OUT. Dire au thread principal du père de dormir pour l'instant.
5. Symétriser le modèle :
   a) Distinguer les tubes utilisés pour envoyer nos demandes (OUTREQ), recevoir les réponses à nos demandes (INRESP), recevoir les demandes qu'on reçoit de l'autre processus (INREQ) et envoyer les réponses à ses demandes (OUTRESP). Notre OUTREQ est donc connecté au INREQ de l'autre processus, et son OUTRESP est connecté à notre INRESP.
   b) Mettre un thread dans chaque processus pour se charger des demandes arrivant le réseau. Le thread gère donc INREQ et OUTRESP tandis que le traitant utilise OUTREQ et INRESP.
   c) Laisser d'abord le thread principal du père dormir et vérifier que les accès du fils (de son thread principal, donc) continuent à bien rapatrier les pages du père.
   d) Modifier ensuite le thread principal du père pour également accéder au tableau. Montrer avec des messages de debug que les pages font bien du ping-pong entre les deux processus.
6. Ajouter du partage de page en lecture seule et montrer que ces pages ne font plus de ping-pong quand ce n'est plus nécessaire.
7. Remplacer le tube par une socket et permettre à plus de 2 processus de partager le tableau.