Compte rendu : Nachos entrées / sorties

I. Bilan

Nous avons implémenté tout le travail obligatoire demandé. Tous les appels systèmes sont fonctionnels et ceux concernant la manipulation de chaînes de caractères peuvent le faire sans couper la chaîne.

Les appels systèmes *getChar* et *PutChar* ne présentent a priori pas de bugs.

Concernant l'implémentation des différentes procédures, *copyStringFromMachine* a été placée dans le fichier *translate.cc*. La procédure est en charge de récupérer une chaine de caractère en mode utilisateur pour la copier en mode noyau.

Cela passe par une traduction d'adresse via la fonction *ReadMem*, il a donc été naturel pour nous de la placer dans le fichier *translate.cc* au sein e la classe *Machine*.

La procédure *copyStringToMachine* est aussi incluse dans ce fichier pour des raisons similaires.

Notre procédure écrit au plus *size* caractères. Lorsqu'un caractère de fin de chaîne est lu, il est copié et la boucle s'arrête.
En fin de boucle, un caractère de fin de chaîne est forcé. Si la boucle se termine à cause d'un caractère de fin de chaîne, il est recopié au même endroit. Sinon, cela veut dire que *size* caractères ont été lu. Comme le buffer a un taille de *size +1*, la dernière case du buffer contient donc le caractère de fin de chaîne. En cas *d'EOF*, un caractère de fin de chaîne est copié pour garantir la stabilité du système. Il n'y a également aucun problème si aucun caractère n'est écrit dans la console.

Nos fonctions réalisent leur but a priori sans bugs et répondent au travail demandé. Certaines d'entre elles pourraient cependant bénéficier d'un peu d'optimisation.

II. Points délicats

Ce qui nous a demandé le plus de temps est la compréhension globale de l'environnement Nachos et le découpage des différentes fonctions et parties du simulateur. Nous avons alors pris le temps d'écrire sur le papier le découpage de Nachos pour bien l'assimiler.Une fois ceci fait, nous avons pu implémenter nos premiers appels systèmes.

Nos buffers locaux servant à stocker les chaînes de caractères sont taille *MAX_STRING_SIZE +1*. Cela permet à l'utilisateur de na pas voir sa chaîne tronquée si sa taille équivaut à *MAX_STRING_SIZE*

Le plus complexe des appels systèmes a été *GetString*. La version basique de l'appel système qui coup la chaîne de caractère a été plutôt rapide à implémenter. La gestion d'une chaine de caractère comprise entre *MAX_STRING_SIZE* et la taille passée par l'utilisateur nous as demandé beaucoup de travail.

Nous avions tenté d'utiliser un compteur pour le nombre de caractères lu mais nous n'avions pas différencié celui ci des caractères restants à lire. Notre schéma de boucle n'était pas non plus le plus adéquat. Nous voulions que la boucle s'arrête d'elle même lors de la fin de la lecture (à la manière de *PutString*), mais cela était impossible.

Nous forçons donc la sortie de la boucle de lecture lorsqu'un caractère de fin de chaîne est détecté. Cela équivaut dans notre code à regarder si le nombre de caractères lu est inférieur à *MAX_STRING_SIZE*.

La procédure *copyStringFromMachine* nous a posé un problème de compréhension de codage des informations. En effet, nous avions eu du mal a comprendre que 4 octets sont lus, mais que nous voulons uniquement 1 octet dans le cas ou le caractère n'est pas un entier et prends donc 1 octet mémoire. L'information a récupérer est donc placée sur l'octet final. Le *cast* en char nous permet de récupérer l'information sans perte puisque les autres octets sont composés de 0.

Des problèmes de compréhension de l'assembleur se sont fait ressentir lors de la gestion de la valeur de retour d'un programme. Il nous as fallu bien observer le fichier *start.S* pour nous rendre compte que la valeur de retour n'était stockée dans aucun registre mais aussi que cette valeur était écrasée lors de l'appel système *SCExit*. Nous avons donc sauvegardé cette valeur dans le registre 4. Elle peut donc être utilisée maintenant dans les handlers des appels systèmes.

Il nous a également fallu du temps pour comprendre qu'un bout de code en assembleur était en charge du lancement du programme utilisateur et que par extension, ce bout de code est aussi responsable de la gestion de la valeur de retour.

Les appels systèmes concernant des caractères simples n'ont posé aucun problème particulier.

Limitations

Nos deux appels systèmes utilisant des boucles, *PutString et GetString* en posent aucun problème de fuite mémoire. En effet, le buffer stockant les chaînes de caractères est alloué de façon statique lors de l'initialisation de l'appel système. Une fois l'appel système terminé, il est automatiquement détruit. Le principal inconvénient de cette méthode peut se voir lorsque l'on a un grand nombre d'appel systèmes successifs. À chaque appel, un nouveau buffer est créé. Cela implique donc lecture, écriture et allocation à chaque appel. Pour éviter cela, il aurait peut être été envisageable de s'imposer un buffer global alloué dès la création de la machine. Ainsi les performances des appels systèmes auraient été améliorés. Cependant cette méthode nous garantit que le buffer est bien libéré et ce après chaque appel système.

Nos procédures et méthodes ne gèrent pour l'instant pas les appels concurrents. Nous sommes dans l'incapacité de bloquer les appels si plusieurs processus désirent écrire ou lire des caractères en même temps. Nos fonctions d'écriture et de lectures ne sont pas atomiques. Dans le cas de plusieurs processus exécutant des appels concurrents, des caractères peuvent être lu par le mauvais processus ou écrit à des emplacements mémoires non adéquats.

Nous sommes arrivé à la conclusion que ne nous pouvons pas agir sur les fonctions en elle même pour empêcher cela. Il faudra donc mettre en place plus tard une sécurité autour de cette portion de code critique.

Tests

Nous avons écrits plusieurs programmes de tests permettant de mettre en évidence tout d'abord les limites de notre implémentation.

Par exemple pour l'appel système *GetString* nous avons :

- Un premier programme qui teste plusieurs appels systèmes successifs dont la chaîne de caractère est plus courte que la taille de notre buffer.
- Le deuxième test opère de la même façon mais cette fois-ci les chaîne de caractères sont exactement la taille du buffer.
- Enfin le dernier test opère sur des chaînes plus longues que la taille du buffer pour tester notre boucle et voir si aucun caractère n'a été perdu ou remplacé.

Par la suite nous avons établi un test mêlant tous les appels systèmes sur de grosses chaînes de caractères. De cette façon, la machine est en état de « stress ». Cela nous permet de vérifier que les retours des appels systèmes s'effectuent sans problème et que les changements successifs d'appels systèmes ne posent pas de problème. La machine est ainsi lourdement sollicitée. Comme nos buffets sont alloués statiquement à chaque appel, cela permet également de voir si des allocations et destructions successives n'affectent pas la stabilité de la machine.