Rapport du Devoir 1 : Entrées/Sorties dans NACHOS
De Florian Claisse et Maxime Zielinger

Bilan

La stratégie d'implémentation suivie a été celle donnée par l'ordre des questions. Nous n'avons pas eu de réel choix à faire à part pour l'intégration de "copyStringFromMachine" que nous avons choisi d'implémenter dans le fichier exception.cc car il est le seul fichier à être réellement en lien entre la partie utilisateur et la partie noyau.

Réponses aux questions du document.

Partie 1 :

La sortie raisonnable attendue de la fonction putchar selon l'exemple donné serait "abcd" plus un retour ligne.

Partie 5 :

V.2 : ReadMem demande un pointeur vers un int en value, or, nous avons des pointeurs vers des chars. La différence est qu'un int est sur 4 octets alors qu'un char n'est que sur 1 octet. Il est donc nécessaire de faire manuellement la conversion du int* vers le char*.

V.3 : Il ne serait pas raisonnable d’allouer un buffer de la même taille que la chaîne MIPS car dans le cas où l'utilisateur enverrait une chaîne de caractères gigantesque, cela demanderait d'allouer un buffer de taille similaire ce qui consommerait une quantité de ram non négligeable.

Partie 6 :

VI.1 : Le message d'erreur contient "Unimplemented system call" qui vient du fait qu'enlever le Halt() lance l'appel système SC_Exit qui n'est pas implémenté de base dans le switch de l'ExceptionHandler. Pour empêcher cette erreur, il suffit d'ajouter un case dans le switch de l'ExceptionHandler pour gérer l'appel système exit.

Lire la valeur de retour du main :
Dans le fichier start.s, il existe une fonction __start qui est appelée avant chaque appel de la fonction main afin de lancer le programme. Après l'exécution du main, ce dernier renvoit sa valeur de sortie dans le registre 2. Mais la valeur copiée par exit dans le registre 4 est la valeur du registre 0. Afin de corriger cette erreur, il suffit de modifier la fonction __start pour que la copie vers le registre 4 soit bien la valeur du registre 2.

Partie 7 :

VII.3 : Si plusieurs threads appelaient en même temps la même fonction, le noyau alternerait entre les différentes fonctions. Ces dernières alterneraient et enverraient une réponse une fois terminé. Mais seule la dernière fonction enverrait une réponse qui serait affichée par la machine.

Partie Bonus :

VIII.0 : C'est une mauvaise idée d'avoir un appel printf car nous avons déjà un appel système putString. Nous nous retrouverions avec deux appels système qui font la même chose.
En plus d'alourdir le noyau, cela rajoute des appels systèmes qui ne sont pas nécessaires et qui complexifient d'autant plus le code.

Points délicats

Nous n'avons pas réellement eu de points délicats hormis des erreurs qui relèvent plus de l'inattention que de réelles difficultés.
Cependant, certains détails comme la gestion théorique des threads nous on fait nous interroger sur la façon dont on pourrait les implémenter plus tard. Il ne s'agit que d'hypothèses pour l'instant, ne sachant pas si une certaine méthode sera conseillée plutôt qu'une autre.

Limitations

Nous n'avons, au cours de notre projet, eu qu'une unique limitation à implémenter. Il s'agit de la variable "MAX_STRING_SIZE".
Nous avons choisi de limiter cette dernière à 10. Il s'agit d'un choix arbitraire afin d'empêcher l'utilisateur d'envoyer une chaîne de caractères d'une taille démesurée (et une valeur réduite pour faciliter nos tests).
Cette limite peut être modifiée au besoin, mais avec notre valeur, nous nous assurons normalement qu'il n'y aura aucun problème et ce, peu importe ce que demande l'utilisateur.

Tests

En nous basant sur putchar.c, nous avons ajouté des tests similaires pour getString.c, putString.c, getChar.c, printf.c, afin de nous assurer que tout fonctionne selon nos besoins.

Pour réaliser ces tests, nous avons opté pour deux solutions.
Tout d'abord, et quand l'option était envisageable, nous avons décidé de tester les fonctions par nous même. C'est le cas par exemple de getchar.c qui est testé grâce à un appel à putchar pour nous permettre d'entrer nous-même les caractères à tester en condition "réelles".

La seconde solution, que nous avons choisi lorsque la première n'était pas envisageable est plus simplement de tester les combinaisons qui pourraient poser problème

La limite majeure dans les deux cas étant nos capacités à imaginer des scénarios qui poseraient problème

Un point à noter cependant à propos du test sur PutString.c.
La limitation précédemment explicitée de MAX_STRING_SIZE doit ici être traitée avec une attention particulière. En effet, si une chaîne entrée dans la fonction dépasse la limite de MAX_STRING_SIZE, il nous a fallut vérifier qu'elle s'affiche bien en entier.
De plus, il est important de vérifier que quoiqu'il arrive, la chaîne n'est pas modifiée lors de l'impression à l'écran. Pour ce faire, nous avons testé d'afficher plusieurs fois la même chaîne stockée dans une variable pour vérifier que nous ayons bien le même résultat à chaque fois (le tout séparé de \n pour améliorer la lisibilité et vérifier qu'envoyer un unique caractère ne pose pas de soucis).