Systèmes d'Exploitation

Devoir 1 : Entrées/Sorties dans NACHOS

Noms:	***Numéros étudiants:***
Kerim Canakci	220 189 32
Corentin Drezen	223 094 35

I . Bilan

Nous avons mis en place les appels systèmes qui permettent d'écrire ou de lire des caractères dans la console tel que l'appel *PutString* ou *GetString* qui permettent d'écrire ou de lire une chaîne de caractères, ou encore notre propre fonction utilisateur *printf (monprintf)* en ré-utilisant du code Linux et notre appel *PutString*.

Nous avons également implémenté le 'handler' de l'appel système Exit et modifié cet appel afin de ne pas avoir à utiliser *Halt* et pouvoir obtenir la valeur de retour des programmes et le exitcode.

D'après nos programmes utilisateurs de tests tout semble fonctionner normalement mais on ne peut pas utiliser trop de mémoire et il n'est pas impossible qu'il y ait des pertes de caractères dans nos fonctions qui gèrent les appels d'écriture ou de lecture si par exemple on appelle les fonctions en dépassant la taille maximum autorisée.

II . Points délicats

Comme une de nos machines ne pouvait pas utiliser le cross compiler linux nous avons fait beaucoup de push sur une autre branche pour pouvoir tester nachos en compilant au cremi à distance.

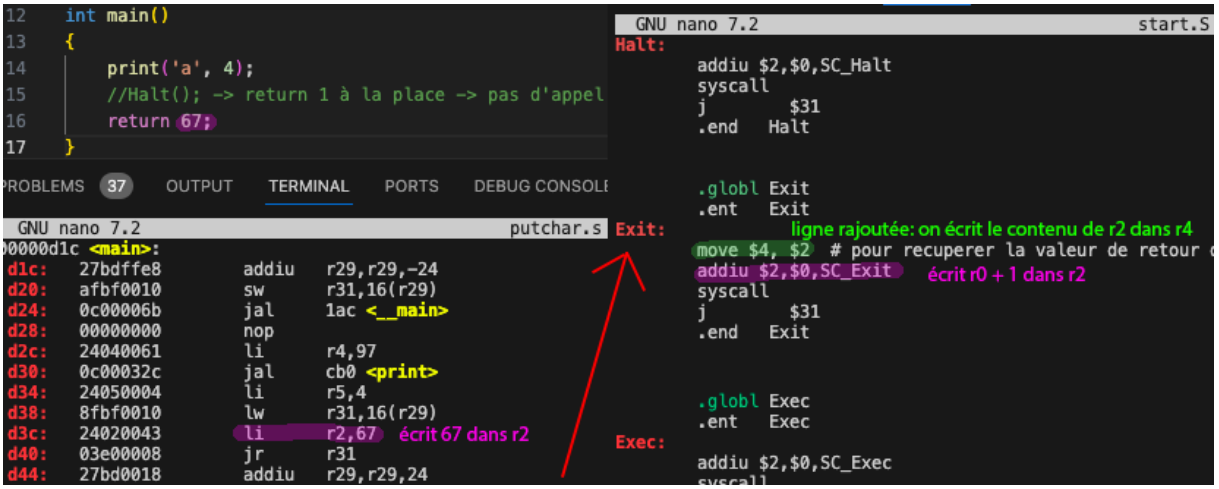
Nous avons eu du mal à comprendre comment implémenter *PutString* et *GetString* car il était difficile de visualiser le cheminement des appels système aux *handlers* et à quoi servirait les méthodes de ConsoleDriver. De plus, nous n'avions pas compris tout de suite dans quels registres lire et nous nous attendions à pouvoir lire ou écrire plusieurs caractères à la fois en mémoire MIPS.

D'autre part, lorsque l'on déclare plusieurs tableaux de caractères à la suite de tailles fixes différentes à l'intérieur du main dans nos programmes test , il semble que les tailles soient mal allouées et provoquent donc des erreurs de mémoire par la suite. La solution à été de déclarer les buffers à l'extérieur du main et de mettre des defines des tailles souhaitées.

De même lorsque l'on essayait d'utiliser environ plus de 2800 octets de mémoire pour nos tableaux, on avait une erreur 'bad_alloc' quand on lançait le programme dans nachos.

En réalisant la partie VI nous avons compris que si on enlevait l'appel *Halt* à la fin d'un programme il y avait une erreur car l'appel *Exit* n'était pas implémenté. Cependant nous avons eu des difficultés à trouver comment renvoyer la valeur de retour d'un programme mais l'énoncé a beaucoup aidé en précisant que l'on pouvait faire la commande *make* avec le nom d'un programme test pour obtenir son code MIPS.

Le code MIPS à été difficile à lire mais en repérant les numéros de registre et en se renseignant sur les instructions nous avons compris que le *main* d'un programme écrivait sa valeur de retour dans le registre 2 mais que l'appel Exit réécrivait par dessus le exitcode. Dans "*Start.s*" nous avons donc réécrit la valeur de retour dans le registre 4 à l'aide de l'instruction *move* avant que le registre 2 ne soit modifié afin de pouvoir la récupérer dans le point d'entrée de du noyau, "*exception.cc*".



Lors de la mise en place du *printf* il a fallu apprendre à lire un Makefile pour pouvoir l'éditer afin d'inclure automatiquement *vsprintf.o* à la liaison des programmes pour pouvoir l'utiliser dans ceux-ci, mais sans l'inclure dans la liste des programmes complets à compiler.

La fonction mon*printf* est définie dans *vsprintf.c* afin de pouvoir être appelé dans tous les programmes utilisateurs sans avoir à inclure un autre fichier.

Il n'y a pas d'appel système *Printf* car ce n'est pas au système de formater le string, cela coûterait du temps de noyau et l'appel pourrait être facilement mal implémenté et détourné et devenir la cause de vulnérabilités.

Dans la fonction PutString nous avons réglé le problème du troncage lorsque MAX_STRING_SIZE est inférieur à la chaîne que nous voulons copier. Pour cela nous avons fait une boucle en copiant la chaîne par paquet de MAX_STRING_SIZE dans exception.cc et qui s'arrête lorsque tout le fichier est lu est copié.

III . Limitations

Dans l'ensemble tout semble fonctionner et il est maintenant possible d'utiliser *printf* et de terminer un programme et de renvoyer une valeur de retour sans utiliser *Halt,* cependant il n'y a pas actuellement de moyen d'utiliser la valeur dans un autre programme car notre implémentation permet simplement d'exécuter un programme, d'afficher sa valeur de retour et d'éteindre la machine.

Il y a un sémaphore pour le PutString et le GetString afin d'empêcher que plusieurs threads puissent écrire ou lire en même temps dans la console.

Nos appels GetInt et PutInt ne permettent pas de gérer des entiers plus grands qu'un *int* mais ils permettent de reconnaître un entier négatif qui prend au maximum 10 caractères. Nous avons dû changer la taille des buffers à 11 (voir 12 pour la terminaison d'un string) au lieu de 10 au cas où il y aurait un '-' au premier caractère et forcé le dernier char à 0 si l'entier est positif.

Notre handler de l'appel système *GetString* utilise la méthode *void ConsoleDriver::GetString(char*, int)* qui effectue des *GetChar* tout en cherchant une terminaison de chaîne de caractères mais sa valeur de retour et ses arguments ne permettent pas de récupérer la nouvelle taille de chaine si elle diffère de celle entrée en argument. *copyStringToMachine* qui va de nouveau chercher si il y a une terminaison plus tôt. On pourrait faire les GetChar directement dans le *handler* afin de vérifier la terminaison sans réitérer et ne pas utiliser *ConsoleDriver::GetString* dans ce handler. On pourrait aussi ne pas effectuer de vérification dans le *copyStringToMachine* si il y'en a eu une précédemment.

IV . Tests

- Pour tester les appels PutString et GetString on a respectivement des programme “*putstring”* et *“getstring”*.

  Pour la fonction PutString nous avons un test si MAX_STRING_SIZE est superieur à la taille de la chaîne que nous voulons copier et pour le cas où nous avons une chaîne de grande taille > 1000.

  Notre programme “*getstring”* fait simplement un appel GetString avec comme argument un tableau de caractères de taille 10 et un entier de valeur 10 comme nombre de caractères à lire. On peut alors taper autant de caractères que l’on veut et appuyer sur entrer. Les caractères reçus sont ensuite écrits un par PutString. On doit obtenir au plus 10 caractères.

  Notre programme “*getstring1”* fait la même chose que le précédent mais avec un tableau de 2000 caractères.

- Les appels PutInt et GetInt peuvent être testés avec le programme “*putint_getint”*. Il appelle GetInt directement et on peut alors entrer un entier signé de 4 octets dans la console qui sera ré-affiché par PutInt. On fait de nouveau un appel PutInt pour afficher un entier qui prend 10 caractères pour vérifier qu’il peut bien reconnaître et afficher le maximum de caractères possible pour un int.

- L’appel *Exit* est testé avec le fichier test “*putchar” où le Halt()* est commenté et le main renvoie 67. Le *handler* va ensuite afficher sa valeur de retour et son exitcode.

- Pour tester le *printf* nous avons un programme test “*printf_test*” qui teste à les format string, décimal et hexadécimal en un appel à *monprintf(char* fmt, …)*, la fonction est définie dans *vsprintf.c*.