université de BORDEAUX

Rapport de Devoir 1

Université de bordeaux

Master informatique 1ère année

Discipline : Informatique

Système d'exploitation NACHOS

Systèmes d'exploitation

Prestat Dimitri
San Nicolas Ludovic

10 octobre 2016

Bilan

Entrées-sorties asynchrones

Nous avons compris grâce aux méthodes `Condole::GetChar` et `Console::PutChar` que la récupération et l'affichage de caractères est une opération qui peut-être longue. Pour éviter de perdre de temps du point de vue du processeur, les opérations de lecture et d'écriture utilisent des sémaphores.

Dans le cas où il était demandé d'entourer chaque caractère lu par les symboles < et >, nous avons choisi de dupliquer le code `console->PutChar()` directement suivi de l'appel à `writeDone->P`. Nous avons pu voir que l'appel à `WriteDoneHandler` pouvait être effectué plusieurs fois dans un même tour de boucle. Nous avions également pensé à créer un tableau avec trois fois plus de place que in dans la fonction `ConsoleTest` puis à copier tous les caractères de `in` dans ce tableau en les entourant de < et > puis à boucler sur ce tableau, mais cette solution aurait été plus coûteuse en mémoire.

Dans le cas de `SynchConsole` les verrous des sémaphores se font dans les fonctions `SynchPutChar` et `SynchGetChar`, il suffit donc d'appeler trois fois ces fonctions pour entourer chaque caractère écrit de < et >.

Communication entre l'utilisateur et le noyau

La mise en place d'un appel système `PutChar` n'a pas été très difficile à comprendre.

— Définir une variable de précompilation dans `syscall.h` du type `SC_ <nom>`

— Déclarer une fonction avec le nom de l'appel système dans `syscall.h`

— Ajouter le code assembleur de cette fonction dans `start.S`

— Ajouter dans le fichier `exception.cc` le code exécuté par la fonction.

Seule l'instruction magique `syscall` nous apparaît un peu floue.

Concernant les chaînes de caractères, les codes ne sont pas si différents, la différence réside dans le fait que le noyau ne donne plus l'adresse d'un caractère mais l'adresse d'une chaîne de caractères, la création d'une fonction `copyStringFromMachine` doit donc permettre de copier le contenu de cette mémoire dans un tableau de caractères classique.

`GetChar` et `GetString` sont faites sur le même principe que les précédentes mais en sens inverse.

`GetInt` et `PutInt` ont été faciles à faire une fois compris le fonctionnement de `GetChar` et `PutChar`, il a simplement fallu convertir le nombre en chaîne de caractères avec `snprintf` ou de chaîne a entier avec `sscanf`.

Nous avons également essayé de mettre en place la fonction `printf` mais nous n'avons pas réussi et elle ne fonctionne pas du tout.

Points délicats

Dans ce devoir nous avons vu pas mal de points délicats.

Lors du passage en mode noyau a la suite d'une exception, nous ne comprenons pas bien comment fonctionne le code. En particulier dans le cas des chaînes de caractères, en effet nous ne voyons pas ou la mémoire à été allouée dans la machine et il est donc difficile de garantir qu'il n'y aura pas de dépassement de tableau en cas d'utilisation de `copyStringToMachine`.

Lors de la lecture de caractères depuis le noyau avec `copyStringFromMachine`, il fallait assurer que tous les caractères présents dans la machine soient bien lus. Si la taille du buffer était trop petite, il fallait donc rappeler la fonction jusqu'à trouver un `\0`. Un problème a été : pour savoir qu'on à a faire a un `\0`, il faut faire un `ReadMem`. Sauf que si le buffer est plein, il faut arrêter de lire dans la machine et retourner la fonction. Mais dans ce cas le dernier caractère lu est perdu et au prochain appel ne sera plus dans la machine. Notre solution consiste à remettre ce caractère dans la machine avec la fonction `WriteMem`.

Lors de l'implémentation de `printf`, nous avons bien créé `vsprintf.c` (qui a d'ailleurs été mis en quarantaine dans le dossier code, en dehors du reste) avec la fonction `vsprintf` et en implémentant les différentes fonction (`isdigit`, `isxdigit...`), mais n'avons pas réussi à modifier le Makefile pour pouvoir différencier le `vsprintf.c` des autres fichiers de test dans le dossier test. Un prototype du Makefile nommé `TMP_ Makefile_ vsprintf` se trouve dans le dossier code, mais ne fonctionnant pas comme souhaité, a été mis en quarantaine pour ne pas gêner la compilation des autres fichiers de test.

Ensuite, un prototype commenté de `printf` se trouve dans `threads/system.cc`, mais comme `copyStringFromMachine` et `copyStringToMachine`, nous n'avons pas trouvé de bon endroit pour cette fonction. De plus, nous ne savons pas si elle doit appeler `putString`, qui est un appel système, dans la fonction, ou bien appeler `copyStringFromMachine` (mais dans ce cas nous ne savons pas quel valeur mettre comme pointeur de registre), ce qui explique aussi le fait que nous ne savons pas où placer cette fonction.

Une dernière incompréhension concernant, l'intérieur de `printf` ainsi que l'emplacement de cette fonction, est que nous n'avons pas très bien compris où l'écrire pour qu'elle soit accessible dans les programmes utilisateurs sans pour autant être un appel système.

Limitations

Notre implémentation actuelle à pour limitations :

— La taille des chaînes de caractères prises en charge par `GetString`, car nous ne savons pas ou trouver la taille du buffer de la machine.

— Les fonctions `copyStringFromMachine` et `copyStringToMachine` ne sont sans doute pas au meilleur endroit mais nous n'avons pas trouvé de meilleure place compte tenu de nos connaissances actuelles dans cette implémentation de NACHOS.

Tests

Pour les programmes de test, nous nous sommes contentés d'écrire des appels systèmes pour tester leurs bon fonctionnement.

Seul le test de `PutString` est différent, il teste le cas d'une chaîne plus longue que la taille du buffer.

Si seul ce programme teste un cas d'erreur c'est parce que nous n'avons pas de cas d'erreur pour les autres. `GetChar`, `PutChar`, `GetInt` et `PutInt` n'ont que des paramètres simples, le compilateur empêchera de leur passer un mauvais paramètre. Dans le cas de `GetString`, nous ne savons pas quelle est la taille du buffer donc nous ne pouvons pas tester si on le dépasse.