université de BORDEAUX

Université de Bordeaux 1

Master 1 Informatique 2018/2019

Devoir Système d'Exploitation

Rapport 1 : NACHOS Entrées/Sorties

16 octobre 2018

Encadré par :

Pr. GUERMOUCHE Abdou

Réalisé par :

- BAKIR Fatima Ezzahra
- JELLITE Oumayma

NACHOS: Entrées/Sorties

Devoir n°1, Master 1 Informatique 2018/2019

L'objectif de ce devoir est de comprendre la manipulation et le fonctionnement d'un système d'exploitation en utilisant un mini-système simulé Nachos pour mettre en place quelques appels systèmes de base en étudiant les parties utilisateurs et noyau.

I. BILAN

Nous allons expliquer brièvement ci-dessous le fonctionnement de chaque partie de ce devoir:

Sur la **partie 2** on a juste travaillé sur la version primitive d'entrées-sorties proposé par Nachos pour faire afficher les caractères sur la console en utilisant la fonction **PutChar** implémenté dans la classe console, pour synchroniser les fonctions de réception et d'affichage d'un caractère, Nachos utilise deux sémaphores. Après, on a effectué le travail demandé en affichant «Au revoir» si l'utilisateur à taper 'control-D' ou 'q', et puis on a réussi à écrire les chevrons encadrant chaque caractère, ensuite lire des caractères depuis le fichier in et les stocker dans le fichier out.

Sur la **partie 3** on a mis en place une console synchrone via la classe **SynchConsole** dont on a complété les fonctions **SynchPutChar** et **SynchGetChar** qui permet d'afficher et d'écrire un caractère tout seule ou encadré par des chevrons sur la console en respectant la synchronisation. Ensuite, on a ajouté dans «**main.cc**» le cas ou argv[2] ='-sc' pour lancer la fonction **SynchConsole**.

C'est dans la **partie 4** que nous avons commencé à implémenter un appel système **Putchar** (pour écrire un caractère) en 3 étapes; Premièrement on a ajouté le numéro de l'appel système et la déclaration du prototype de la fonction dans **Syscall.h.** Deuxièmement on a ajouter le code assembleur dans **Start.s .** Enfin , on a traité cet appel système dans le fichier **exception.cc ,** ce traitement consiste à récupérer un caractère stocké dans le registre 4 en mode utilisateur depuis le mode **MIPS** et le passer à la fonction **SynchPutChar** en utilisant la fonction **ReadRegistre** de la classe machine.

Sur cette **partie 5** que nous avons suivi les mêmes étapes pour l'appel système **PutString** pour l'implémenter, cet appel système permet de gérer les chaînes de caractères. Pour cela on a utilisé une fonction intermédiaire **CopyStringFromMachine** (on a choisi de la mettre dans le fichier **system.cc** car c'est une fonction qui communique avec le système) qui permet de copier une chaîne caractère pas caractère du mode utilisateur vers le mode noyau à l'aide

de la méthode **ReadMem** au lieu d'utiliser directement un pointeur vers une chaîne car on ne le dispose pas en mode noyau.

On a même géré le cas ou la chaîne de caractères est de taille grande.

Sur la **partie 6,** si on supprime l'appel **Halt ()** à la fin de la fonction main de **PutChar.c** un message d'erreur apparaît **« Unimplemented systéme call 1 »** , le numéro '1' correspond au **SC_EXIT** qui n'était pas définie dans le fichier **exception.cc.** Pour corriger cet erreur nous avons ajouté l'appel système **EXIT** en appelant implicitement **Halt().**

Et pour prendre la valeur de retour de la fonction main on a ajouté un affichage dans cet appel système qui permet d'afficher cette valeur stockée dans le registre 4.

Concernant la **partie 7,** tout d'abord nous avons implémenté l'appel système **GetChar** pour le faire on a récupéré la valeur de retour stockée dans le registre 2 en utilisant la fonction **Write-Registre**.

Ensuite, nous avons complété la méthode **SynchGetString** de la classe **SynchConsole** pour implémenter l'appel système **GetString**, pour cela nous avons définie une fonction **copyStringToMachine** qui permet de copier la chaîne du caractère du monde noyau vers le monde utilisateur en utilisant la fonction **WriteMem**.

On a pas pu prendre en compte les appels concurrents, nous avons compris que si on appelle deux threads en même temps la chaîne peut être lu dans le mauvais ordre. On peut résoudre ce problème en utilisant les sémaphores sauf qu'on a pas su comment le faire.

BONUS: PutInt/GetInt

De même façons que les autres appels système, on a implémenté le **PutInt** qui écrit un entier signé en utilisant la fonction **snprintf** pour obtenir l'écriture externe décimale. Nous avons fixé un **MAX_INT_SIZE** en 11 car c'est la taille maximale des entiers signés écrits sous forme de texte.

Pour le **GetInt** on a réussi à utiliser la fonction **sscanf** pour lire l'entier saisi par l'utilisateur.

II. Points Délicats:

Au début, nous avons trouvé une difficulté à comprendre la notion des registres, nous avons résolu ce problème juste en lisant les commentaires.

Ensuite, nous avons trouvé des difficultés au niveau des deux fonctions: **CoypStringFromMachine** et **CopyStringToMachine**, car au début on a trouvé du mal à comprendre les deux fonctions **ReadMem** et **WriteMem**, après nous avons rendu compte que ces dernières servent à lire et écrire dans la mémoire virtuelle.

Puis, nous avons pris le plus de temps pour gérer les chaînes de caractère supérieures à la taille **MAX_STRING_SIZE**. Pour le cas de **PutString** on a proposé un algorithme ou on a itéré la taille récupérer par la fonction **CopyStringFromMachine** jusqu'à qu'elle soit différente de la taille Maximale **MAX_STRING_SIZE** déjà fixée au début c'est-à-dire qu'on est arrivée à la fin de la chaîne.

Nous étions bloquées au niveau de la valeur de retour de la fonction main, après nous avons compris que cette valeur est stockée dans le registre $2 c'est_à_dire on doit modifier l'instruction **move $4,$0** par **move $4,$2** pour que la valeur de retour soit dans le registre $4 dans le fichier **Start.s** au niveau de **__Start** qui permet d'initialiser le programme c de test en mode utilisateur.

III. Limitations:

Comme limitation de notre code, dans la partie **getInt** on a pas géré le cas ou notre entier dépasse le **MAX_INT_SIZE**.

Ainsi que nous avons pas eu le temps pour faire la partie BONUS «**un printf** ».

*Parmi les avantages de notre cote c'est le fait qu'on a essayé d'utiliser des noms de variables significatifs et quelques commentaires pour que le code soit lisible.Ainsi, que toute modification est encadrée avec **#ifdef CHANGED … #endif.***

IV. Tests:

Au début, on a examiné **ConsoleTest** en lançant la commande : **./nachos -c** qui permet de tester les appels systèmes **PutChar** et **GetChar** pour la partie Entrées/Sorties Asynchrones.

Pour le test de la partie synchrone on a testé tous les appels systèmes en essayant de vérifier tous les cas possibles dans les fichiers tests, puis on exécute nos tests avec la commande : **./nachos -x ../test/'nom du test'** depuis le fichier **userprog**,

- **PutChar**: On a juste exécuté le code déjà fourni.
- **GetChar**: On a testé s'il affiche correctement le caractère fourni par l'utilisateur.
- **PutString**: On a vérifié le cas de chaîne trop longue.
- **GetString** : On a proposé un test qui permet de récupérer et afficher une chaîne trop longue.
- **PutInt** : On a réussit à afficher un entier avec cet appel système.

- **GetInt** : Ici, on a testé l'affichage de l'entier saisit par l'utilisateur en appelant **PutInt**.
- **Halt**: On a regardé les différents cas de sorties était bien gérer avec et sans **Halt** et quand on retourne une valeur dans la fonction main.