université de BORDEAUX

Projet Semestre 7

Systèmes d'exploitation — Devoir 1

Auteurs :
Ayoub LESFER
Wassim BERRARI
Camille AUSSIGNAC

Professeur :
Abdou GUERMOUCHE

15 Octobre 2018

Sommaire

1 Bilan

L'objectif de ce devoir encadré est de mettre en place sous Nachos un mécanisme d'entrée-sortie minimal, permettant d'afficher ou de récupérer des chaînes caractères en utilisant des appels systèmes.

Les fonctions de gestion d'entrée-sortie des chaînes de caractères fonctionnent, ainsi toutes les parties demandées au devoir ont été effectuées à l'exception de la partie VIII (Bonus).

2 Points délicats

Les points délicats ont été principalement rencontrées en implémentant les appels système PutString et GetString. En effet, il s'agissait de bien manipuler le buffer afin de gérer les débordements de chaînes de caractères.

Dans le cas de PutString, pour afficher une chaîne caractère de taille supérieure à celle du buffer (MAX_STRING_SIZE), nous utilisons un curseur qui pointe sur l'endroit de lecture courant. À chaque itération de la boucle de lecture, le curseur est incrémenté de MAX_STRING_SIZE si la taille de la chaîne de caractère à lire est toujours supérieure à MAX_STRING_SIZE.

Dans le cas de GetString, un processus similaire est exécuté en addition de la variable string_size qui décrit la taille de la chaîne à caractère restante à récupérer.

3 Limitations

Pour les appels système PutInt et GetInt, nous avons choisi une taille maximale (MAX_INT_SIZE 10) qui correspond au plus grand (plus petit) nombre entier signé $2^{31} - 1 (-2^{31})$. Cependant nous pensons qu'il y aurait sûrement une meilleure méthode afin de gérer des nombres plus grands.

4 Tests

4.1 Console progtest.cc

Commande : ./userprog/nachos -c
Affichage :
$ a
<a>

$ aze
<a><z><e>

On utilise ici des sémaphores afin d'assurer la cohérence de données d'écriture et de lecture. On affiche chaque caractère encadré par des chevrons, et nous arrêtons dès que le caractère \n est lu.

4.2 SynchConsole progtest.cc

Commande : ./userprog/nachos -sc
Affichage :
$ a
<a>

$ aze
<a><z><e>

Ici la cohérence de données assurée par les sémaphores est implémentée directement dans la définition de la structure SynchConsole. Nous avons juste affiché chaque caractère encadrés de chevrons jusqu'à la lecture d'un \n.

4.3 putchar.c

Nous avons utilisé une boucle incrémentant la valeur de l'argument de PutChar afin de tester si le caractère lu est bien le bon et s'il écrit bien les caractères suivants. L'exécution de la commande affichera donc a b c d.

4.4 putstring.c

Afin de tester cet appel système, il fallait tester les cas de stress suivants :
— MAX_STRING_SIZE

— MAX_STRING_SIZE + 1
— MAX_STRING_SIZE*2
— MAX_STRING_SIZE*2 + 1

4.5 getchar.c

Pour tester cet appel système, il suffit d'écrire un caractère en entrée de la console. Ce caractère sera ensuite affiché en utilisant putchar

4.6 getstring.c

Pour tester cet appel système, les chaînes de caractère qui correspondent au cas de stress ont été saisi à l'entrée de la console et sont ensuite affichées en utilisant putstring.
— MAX_STRING_SIZE
— MAX_STRING_SIZE+1
— MAX_STRING_SIZE*2
— MAX_STRING_SIZE*2 + 1

4.7 putint.c

Les cas de stress correspondant à cet appel système sont :
— $2^{31} - 1$
— 2^{31}
— -2^{31}
— $-2^{31} - 1$

4.8 getint.c

Les mêmes de cas de stress correspondant à putint ont été saisi à l'entrée de la console et sont ensuite affichés en utilisant putint