Systèmes d'Exploitation

Fiche 1 : Entrées/Sorties dans NACHOS

Laroque Austin & Malmgren Axel

I. Bilan

Nous avons implémenté, sous Nachos, un chemin d'entrées/sorties complet permettant l'exécution de programmes utilisateurs simples, des parties I à VIII du sujet.

- **Console asynchrone** : nous avons étudié le fonctionnement des interruptions (`ReadAvailHandler` et `WriteDoneHandler`) et utilisé les sémaphores (`readAvail` et `writeDone`) pour les synchroniser.

- **Driver synchrone `ConsoleDriver`** : fournit des fonctions de haut niveau (`PutChar`, `GetChar`, `PutString` et `GetString`) qui sont bloquantes le temps que l'opération de I/O se termine, sans attendre en boucle.

- **Appels système** : mise en place de stubs MIPS dans `start.S` et gestion dans `exception.cc` pour `PutChar`, `PutString`, `GetChar`, `GetString` et `Exit(int)`.

- **Fin de programme sans `Halt()`** : le stub `start.S` propage le retour de `main($v0)` vers `Exit(status)($a0)`. Le noyau termine alors correctement le thread utilisateur.

- **Copies sûres MIPS ↔ kernel**: nous avons implémenté `copyStringFromMachine` et `copyStringToMachine`, limitant la copie à la taille d'un tampon noyau et forçant un '\0' final, pour éviter un dépassement.

- **Printf utilisateur** : nous avons réutilisé `vsprintf` du kernel Linux, adapté en implémentant les fonctions manquantes (`isdigit`, `isxdigit`, `islower`, `toupper`, `strnlen`). Nous avons ensuite construit un `printf` utilisateur dans `test/`, qui utilise `vsprintf` et `PutString`. Nous n'en avons pas fait un appel système car cela aurait été coûteux en changements de contexte (User-Kernel) et ce n'est pas vraiment le rôle du noyau de formater du texte, qui lui doit fournir des primitives simples..

Dans l'ensemble, toutes les parties demandées ont été réalisées avec succès. Les fonctionnalités principales fonctionnent correctement normalement.

II. Points délicats

Asynchronisme matériel → synchronisme logiciel

La console Nachos est asynchrone : on ne peut ni lire un caractère avant sa disponibilité, ni écrire un nouveau caractère tant que l'écriture précédente n'est pas finie. Nous avons résolu cela dans le `ConsoleDriver` : pour le `PutChar` on envoie `TX(ch)` puis on bloque sur `writeDone->P()`. Pour le `GetChar` en revanche, on bloque sur `readAvail->P()`, et ensuite on effectue `RX()`.

Sécurité mémoire lors des copies

`ReadMem` et `WriteMem` utilisent des pointeurs vers `int`. Pour éviter tout dépassement, nous avons copié octet par octet dans des tampons noyau bornés et forcé un '\0' à la fin. Cela protège le noyau contre des chaînes utilisateur trop longues ou des pointeurs invalides. Même principe pour la copie contexte noyau vers contexte utilisateur.

Conventions MIPS et UpdatePC()

- Numéro de syscall en `$r2`, arguments en `$r4`, retour en `$r2`.
- Appel à `UpdatePC()` après chaque syscall pour éviter la réexécution.
- À la fin de `main`, ajout de `move $a0,$v0` avant `Exit(status)` pour propager le code retour.

Printf utilisateur

Le point délicat a été l'intégration de `vsprintf` du kernel Linux dans Nachos vu qu'il n'y a pas de librairie C standard. Nous avons ré-implémenté les fonctions minimales nécessaires et fait la liaison dans le Makefile qui a nécessité d'exclure `vsprintf` de la liste des exécutables tout en l'ajoutant automatiquement à la compilation de chaque programme utilisateur.

III. Limitations

Performance (I/O non tamponnées) : actuellement, `PutString` émet un caractère à la fois, ce qui déclenche une interruption par caractère. Une amélioration possible serait de bufferiser côté noyau pour émettre des blocs.

Taille fixe des tampons : le noyau utilise des buffers bornés (`MAX_STRING_SIZE`, buffer interne de `printf`). Des chaînes plus longues sont tronquées.

Pas de gestion multi-threads utilisateurs : dans l'état, nos fonctions sont sûres en séquentiel mais n'ont pas été testées en concurrence.

Printf limité : `printf` supporte les formats de base (%d, %x, %s, %c) mais pas toutes les variantes possible de la librairie C standard.

IV. Tests

Méthode : petits programmes ciblés, commentés (commande & sortie attendue), tests clavier et fichiers (*in/out*), logs *-d s* pour suivre les *case* de *exception.cc*.

Console asynchrone *(./nachos -c [in out])* : écho <*x*>, sortie sur 'q', EOF (Ctrl+D).

Driver synchrone *(./nachos -sc [in out])* : mêmes scénarios via *ConsoleDriver*.

PutChar/PutString *(test/putstring)* : message + séquence (~300 caractères) ; en *-d s*, on voit la suite de *PutChar*.

Exit(status) *(test/ret)* : *main* retourne une valeur (ex. 20) ; log Exit(20) prouve que *move $4,$2* puis Exit fonctionnent.

Lecture (test/getstring) : saisie d'une ligne (conserve *'\n'*), ré-émission ; cas limites : tampon court, ligne > n−1, EOF immédiat.

Printf *:* simple programme dans un main qui utilise le printf implémenté, utilisé de manière semblable à son implémentation dans la librairie standard C. Deux lignes affichées.