Rapport de projet NachOS

Équipe H

Mohd Thaqif ABDULLAH HASIM
Florian BARROIS
Cédric GARCIA
Hosseim NAHAL
Peio RIGAUX

18 janvier 2018

Présentation de notre nom badass

NOM BADASSest un système d'exploitation basé sur le fonctionnement du système Unix. Il propose ainsi une version simplifiée des fonctionnalités de ce dernier, à savoir :

- Un système synchronisé d'entrées/sorties ;
- La gestion de plusieurs processus utilisateurs multithreadés SI ON ARRIVE A FAIRE FONCTIONNER LES TESTS !
- Un système de fichiers permettant la manipulation de fichiers et la navigation à travers les répertoires GERANT LES FICHIERS OUVERTS ? ?et dont la taille des fichiers peut atteindre 112,5Ko.
- La possibilité de communiquer en réseau...DETAILS

Spécifications

1 Entrées/Sorties

char GetChar() :
Retourne le caractère lu sur l'entrée standard.

void PutChar(char c) :
Écrit le caractère c sur la sortie standard.

void GetString(char* s, int n) FONCTION DE LA MORT ! GROS BORDEL Récupère n caractères depuis l'entrée standard et les stocke dans la variable s. Si le caractère de saut de ligne '\n' se trouve dans la chaîne à lire, alors seuls les caractères lus jusqu'ici, '\n' exclus, sont stockés dans s. Les caractères restants sont stockés dans

n premiers caractères est le caractère de saut de ligne '\n', alors la lecture s'arrête et s contient tous les le reste de la chaîne est ignorélecture s'arrête lorsqu'un '\n' ou EOF est rencontré. '\0' est stocké après le dernier caractère dans le tampon s. '\0' est stocké à l'indice 0 du tampon quand la fin du fichier est détectée alors qu'aucun caractère n'a été lu.

void PutString(const char s*)

Parcourt la chaîne de caractères s et écrit caractère par caractère sur la sortie définie précédemment jusqu'à la rencontre du caractère '\0' ou jusqu'à ce que MAX_STRING_SIZE soit atteinte. Le comportement est indéfini dans le cas où s ne contient pas de '\0'.

void GetInt(int* n)

Lit un entier depuis l'entrée standard, si la valeur n'est pas dans $[-2^{31}; 2^{31}-1]$ elle sera forcée à une de ces valeurs (la limite est de 16 caractères pour

les entiers, au-delà les caractères ne seront pas comptés). SI un utilisateur entre le signe '-' sans le faire suivre de chiffres, la valeur 0 est stockée.

void PutInt(int n)

Récupère la valeur de l'entrée définie précédemment puis l'écrit à l'adresse pointée par n. Le comportement est indéfini si la valeur n'est pas dans $[-2^{31}; 2^{31} - 1]$.

2 Threads, processus et synchronisation

int UserThreadCreate(int f, int arg) :

Initialise un thread utilisateur qui appelle la fonction f avec l'argument arg. Bien que UserThreadCreate autorise un seul argument pour la fonction f, il est possible de lui transmettre plusieurs arguments en les définissant comme paramètres d'une structure. Retourne l'identifiant du thread créé.

void UserThreadJoin(int tid) :

Interrompt l'exécution du thread appelant jusqu'à la terminaison du thread identifié par tid.

void UserThreadExit() :

Termine l'exécution du thread appelant. L'appel à UserThreadExit() est facultatif puisqu'il est systématiquement réalisé lorsqu'un thread a terminé l'exécution de la fonction qui lui a été attribuée. Il peut néanmoins être utilisé pour forcer l'arrêt du thread avant la fin de sa tâche.

void Sem_init(Semaphore* sem, int val) :

Initialise la valeur du sémaphore sem à val.

void Sem_wait(Semaphore sem) :

Décrémente la valeur du sémaphore sem. Sem_wait est bloquant tant que la valeur de sem est égale à zéro, empêchant ainsi la décrémentation. L'appel à Sem_wait nécessite que le sémaphore sem ait été initialisé avec la fonction Sem_init.

void Sem_post(Semaphore sem) :

Incrémente la valeur du sémaphore sem. Si, une fois incrémentée, la valeur de sem vaut un alors qu'un fil d'exécution est bloqué par Sem_wait sur le

même sémaphore sem, alors ce fil d'exécution reprend son activité. L'appel à Sem_post nécessite que le sémaphore sem ait été initialisé avec la fonction Sem_init.

void Sem_destroy(Semaphore sem) :
Détruit le sémaphore sem. Si sem est détruit alors que des fils d'exécution ont été suspendus par un appel à Sem_wait sur ce même sémaphore sem, ces fils d'exécution ne se termineront pas avant l'arrêt complet du système. Potentiellement à tester mais devrait être le comportement logique

void ForkExec(char* filename) :
Crée un nouveau processus qui exécute le programme passé en paramètre. Á sa création, le nouveau processus contient un seul fil d'exécution. Les valeurs des registres de ce fil d'exécution sont identiques à celles des registres du fil d'exécution qui a appelé ForkExec.

3 Système de fichiers

Les commandes du système de fichiers doivent être exécutées de la manière suivante :
./nachos-filesys command
Options disponibles :

- -l : Liste le contenu du répertoire courant (équivalent de ls sur Unix).
- -D : Affiche le contenu du disque et les blocs occupés utilisés par chaque fichier.
- -cp (nom1) (nom2) : Copie le fichier de nom nom1 dans le système de fichiers Nachos, sous le nom nom2.
- -p (nom) : Affiche le contenu du fichier nom. Échoue si le fichier n'existe pas.
- -md (nom) : Crée un répertoire de nom nom. Échoue si le répertoire existe déjà ou si l'espace disque est insuffisant.
- -rd (nom) : Supprime le répertoire de nom nom. Échoue si le répertoire n'existe pas, n'est pas vide, ou s'il s'agit du répertoire . ou ..
- -cd (nom) : Se déplace sur le répertoire de nom nom qui devient le répertoire courant. Fonctionne avec un chemin relatif. Échoue si le répertoire n'existe pas.

Pour la création de fichiers, la taille maximale d'un fichier est de 30*30*128 octets soit 112,5ko.

4 Réseau

PAS DE RESEAU PUISQUE PAS DU COTE USER ?

Implémentation

1 Entrées/Sorties

2 Threads, processus et pagination

Un thread contient des champs entiers TID et BID.

Le BID correspond à l'indice du BitMap auquel le thread est référencé. Il est propre à l'espace d'adressage. Plusieurs threads peuvent donc posséder la même valeur de BID s'ils ne font pas partie du même processus. La gestion des threads est ainsi effectuée grâce au TID qui lui est unique dans le système.

L'implémentation des processus est différente de celle d'Unix. Dans NOM BADASS, il n'existe pas de "lien de parenté" entre les processus. En conséquence, par exemple, un processus p2 créé suite à un ForkExec effectué par un thread d'un processus p1 peut appeler UserThreadJoin sur p1 et attendre qu'il termine, pour ensuite continuer son exécution normalement.

La pile est gérée de la manière suivante : Un objet BitMap est utilisé pour déterminer le secteur de la pile lié au nouveau thread et définir le numéro BID de ce thread.

Un type Semaphore a été défini pour procéder à l'implémentation des sémaphores utilisateur. La gestion du UserThreadJoin a été réalisée grâce à l'utilisation d'une liste chainée contenant un couple <TID,Semaphore>.

Enfin, un nouvel argument pointant sur la fonction UserThreadExit a été ajouté dans le fichier start.S lors de l'appel à UserThreadCreate. Ceci permet ainsi la terminaison automatique des threads lorsque ceux-ci ont terminé la tâche qui leur a été assignée.

3 Pagination

Stratégies d'allocation des pages physiques géré par un tirage aléatoire via la méthode « GetEmptyFrame » . Utilisation d'une bitmap pour vérifier si la frame est déjà allouer ou non.WHAT ?

La fins des processus et des threads est gérée automatiquement via un compteur pour déterminer s'il est nécessaire d'effectuer une interruption machine ou une terminaison de thread.

4 Système de fichiers

Pour permettre la création de répertoires, il a fallu distinguer les répertoires des fichiers "normaux". Ainsi dans la classe DirectoryEntry (qui représente une entrée de répertoire, quelle qu'elle soit) on ajoute un champ "isFile" de type booléen. Un booléen a été ajouté en paramètre des fonctions Find, Add et Remove de la classe Directory qui sont notamment utilisées lors de la création et suppression de fichiers ou de répertoires. De plus, un répertoire racine, ne contenant qu'une entrée spéciale "." doit être créé, et défini comme répertoire courant au formatage.

bool MakeDir(const char *name) : de la même manière que pour la fonction Create, cette fonction prend une chaîne "name" en paramètre. On charge le répertoire courant à partir du secteur Directory Sector (constante fixée à 1) dans un objet de type Directory. Un test a été mis en place afin de s'assurer qu'un répertoire de nom "name" n'existe pas déjà (auquel cas on renvoie faux). On charge également le BitMap qui nous donne un secteur de données disponible (s'il n'y en a pas, on renvoie faux également). Un nouveau répertoire est crée. Il contient deux entrées spéciales (. et ..) et le FileHeader associé, qui est écrit sur le secteur libre défini précédemment. Enfin sur le répertoire courant une nouvelle entrée qui redirige vers ce secteur est ajoutée, et toutes ces modifications sont réécrites sur le disque.

bool ChangeDir(const char *name) : Cette fonction va chercher le répertoire dont le nom (ou le chemin relatif) est passé en paramètre et, s'il existe, en faire la copie sur le répertoire courant situé sur le secteur DirectorySector = 1.

bool RemoveDir(const char *name) : Si le répertoire dont le nom est passé en paramètre se trouve sur le répertoire courant (sauf . et ..), on

vérifie si le répertoire est vide et, le cas échéant, on libère tous les secteurs qu'il utilise et le supprime des entrées du répertoire courant, puis on valide les modifications en réécrivant le répertoire courant sur le disque.

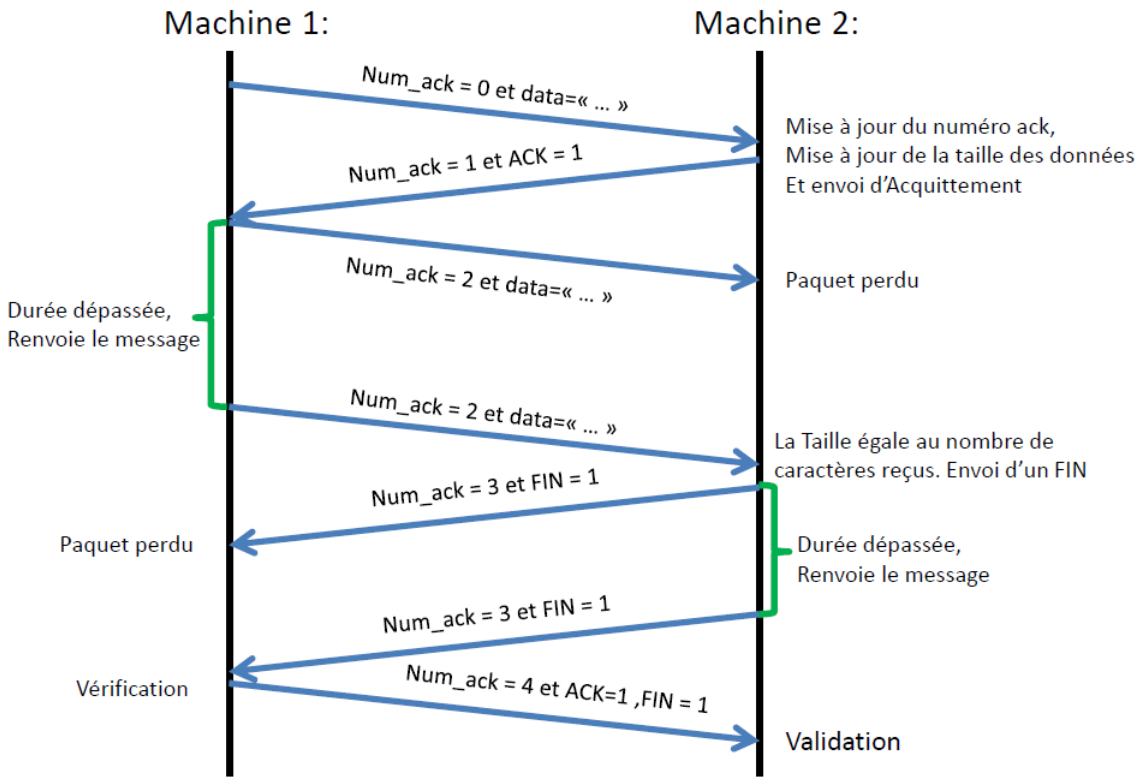
Extensions possibles : -Ajout d'une option "-mv (nom) (rep)"" [Déplacer un fichier ou un répertoire de nom nom vers le répertoire rep. - Extension de l'option "-cp (nom1) (nom2) : Effectuer une copie d'un fichier nom1 présent dans le système vers un nouveau fichier nom2 également dans le système.

-En utilisant une classe similaire à FileHeader mais permettant le stockage de 32 entiers (soit 128 octets) au lieu de 30, on peut étendre la taille maximale d'un fichier à 30*32*128 octets soit 120ko.

-Permettre l'utilisation de chemins pour le mkdir et le rmdir.

5 Réseau

La partie réseau du système fonctionne grâce à un protocole de communication qui a été créé en se basant sur le fonctionnement du protocole TCP/IP. L'échange d'information s'effectue comme indiqué ci-dessous.



Les longs messages dépassant la taille d'une trame sont gérés par un séquencement et l'ajout d'une taille totale dans l'en-tête de la classe MailHeader.

Les messages sont vérifiés grâce à leur numéro d'acquittement et par leur type ACK, FIN ou donnée, ceci dans le but de conserver l'ordre des messages et d'assurer une commmunication cohérente entre le récepteur et l'émetteur.

Tests utilisateur

1 Test sur la console :

Pour l'instant, les tests fonctionnent à l'aide de l'entrée/sortie standard et les types de données pouvant être entrés sont testés manuellement (tentative de dépasser la taille que peut stocker un int pour getint.c par exemple).

getchar.c : Récupère un caractère sur l'entrée puis l'affiche, lui et les deux caractères le suivant dans l'ordre alphabétique. getstring.c : Demande le nom de l'utilisateur (une chaîne de caractères) puis l'affiche précédé de « Bonjour Monsieur ». getint.c : Demande un entier à l'utilisateur, le borne afin de le faire rentrer dans un int, puis l'affiche (Ce test sert à la fois pour SynchGetInt() que pour SynchPutInt()).

Organisation du travail en équipe

Répartition des tâches

Peio : Compte rendu et exécution des tests. Les tâches assignées à chacun montrent l'organisation générale du groupe. Cependant, toute l'équipe se tient au courant de ce que chacun fait et peut ponctuellement travailler sur la même tâche (par exemple lors de la rédaction du premier compte rendu et de la compréhension du fonctionnement des piles de thread NachOS).

Retour global sur le projet

[à remplir à la fin]