I . Bilan

Nous avons implémenté le traitement EOF dans la fonction consoleTest, l'ajout des fonctions synchrones putChar et getChar.

-La fonction getChar attend qu'un caractère soit envoyé puis le lis et le retourne.

-La fonction putChar quant à elle écrit le charactère en validant directement l'écriture (ré-initialise à false la variable booléenne qui vérifie que seulement une personne écrit à la fois), cela permet une écriture synchrone. La validation du traitement est gérée dans la fonction.

-Nous avons essayé les fonctions ConsoleTest en lui passant des arguments in (fichier déjà crée) et out.

-La fonction GetString, il se peut qu'il y ait des bugs que l'on ne pas résolu. Par exemple si l'utilisateur envoie certains caractères ils risquent de ne pas être pris en compte

-La fonction PutString, Nous n'avons pas trouvé de bug, Les tests ont été faits sur des grosses chaînes,des petites chaînes avec un buffer de taille petite/de taille moyenne/ de grande taille.

-La Fonction PutInt, l'utilisateur envoie autre chose qu'un entier, il se peut qu'il y ait des problèmes dans l'écriture en mémoire dans la machine. Il faut que l'utilisateur suive à la lettre la consigne.

-La fonction GetInt : récupère un entier envoyé par l'utilisateur, si l'utilisateur passe une valeur supérieure à MAX_INT, la fonction redemandera de le retaper. Il y a des bugs d'écritures pour les nombres négatifs nous avons essayé de les corriger

-La fonction copyStringToMachine écrit dans l'espace mémoire de la machine à l'emplacement de la première variable passé en paramètre (registre 4), dans ma boucle il écrit tant que WriteMem renvoie vrai et qu'il n'a pas rencontré de \0 qui signifie le fin de la chaine ou que la taille écrite ne soit pas supérieure à la taille envoyé en paramètre sinon entraine une erreur sur l'espace mémoire machine sinon : Invalid write of size X

-La fonction copyStringFromMachine récupère les éléments dans la mémoire pour les placer dans la chaine de charactère (to). Comme copyStringtoMachine j'ai utilisé une boucle while avec les même conditions d'arrêts pour éviter les débordements.

-La récupération de la valeur de retour a été implémenté de sorte à ce qu'elle soit placée dans le 3 ème registre car il s'agit d'un registre inutilisé. Il peut être utile à l'avenir.

-Nous avons implémenté un sémaphore dans exception.cc, pour qu'un processus mette en attente les autres processus jusqu'à ce qu'il termine au niveau noyau.Il permet aussi d'éviter que plusieurs processus écrivent/lisent dans le même emplacement mémoire à la fois

-Nous aurions très bien pu le mettre dans consoleDriver mais nous estimons qu'il est plus pertinent dans exception.cc, chaque thread exécutera chaque fonction de la librairie un par un plutôt que tous en même temps et puis c'est un sémaphore qui protège le noyau de la machine.

II . Points délicats

Les deux points délicats à implémenter sont PutString et GetString:

Pour PutString, Il y a plusieurs cas qui posent problème.

- Soit l'utilisateur a rentré une chaine en paramètre plus petite que la taille du buffer de la machine, dans ce cas il suffit de récupérer seulement les éléments de la chaine de charactère en mémoire et de les afficher dans la console à l'aide de la fonction PutChar, charactère par charactère. Pas besoin de boucle.

- Dans l'autre cas, Il faut lire la chaine tant que l'on a lu autant de charactère que la taille de la buffer, chaque itération correspond à une lecture entière du buffer pour l'afficher dans la console, le dernier élément étant un \0. Ce \0 est nécessaire sinon il entraine la lecture d'une case mémoire de trop. Donc on ne lit que MAX_SIZE_BUFFER-1 par itération. Il faut aussi penser à garder l'emplacement (track) de la case mémoire sur laquelle on lit. Il faut aussi vérifier que la taille lu en machine soit inférieur à la taille du buffer pour arrêter la boucle

Pour GetString, Il y a plusieurs cas qui posent problème comme pour PutString.

- Soit L'argument n passé en paramètre est inférieur à la taille du buffer. Il suffit simplement de faire une boucle de GetChar et qui va placer un charactère par un dans la chaine de charactère. Ils seront ainsi écrit en mémoire jusqu'à la énième case demandée sans faire de boucle dans exception.cc.

-Soit n est supérieur au buffer dans ce cas, il faut récupérer size du buffer -1 caractères. Lorsque qu'il reste moins de caractères à récupérer que la taille du buffer il faut simplement récupérer que le nombre restant de caractères.

Pour que le programme se termine en récupérant la valeur de retour, on passe la valeur au registre du registre 2 (là où normalement la valeur de retour est enregistré puis écrasé par SC_EXIT) vers le registre 3 (non utilisé). Ce fut un point assez délicat car assez compliqué à comprendre il fallait compiler un fichier .c en .s pour savoir vers dans quel registre il envoyait la valeur de retour ensuite modifier dans Start.S pour qu’il le passe au registre 3.

III . Limitations

Dans la fonction GetInt, l'utilisateur ne peut que taper des entiers positifs, les valeurs négatives entrainent une erreur écriture sur une mauvaise adresse mémoire. Je pense que l'erreur vient de la taille du buffer et de n.

En parlant de la taille du buffer, pour PutInt nous avons implémenté une taille de 26, taille suffisemment grande à mon sens pour écrire un entier. Le plus gros entier possède 10 chiffres soit 10 cases, plus 6 cases pour l'afficher sous forme de flottant. Normalement il n'est pas possible de dépasser cette taille sinon ce n'est plus un entier mais un long, ou un double …

Mon implémentation de GetInt n'est pas satisfaisante, la taille du buffer n'est pas convenable et l'argument n envoyé entraine des problèmes certains pour un nombre négatif.

IV . Tests

Pour chaque fonction implémentée, il a fallu faire des programmes de tests afin de vérifier que celles-ci fonctionnent de la manière voulue. Nous avons testé différents paramètres afin de vérifier le maximum de conditions.

PutChar:

La fonction PutChar étant relativement simple, il suffit de passer un code entier d'un caractère tout en restant dans la limite des codes ascii afin de la tester correctement. La fonction "print" donnée dans le sujet permet de tester la fonction correctement, PutChar fonctionne correctement sans problème et affiche les caractères l'un après l'autre.

De même, en faisant un appel unique avec un code entier arbitrairement choisi, le bon caractère est affiché dans la console.

Si on teste avec une valeur très grande, la fonction n'affiche rien à partir des entiers ne correspondant plus à des caractères ascii. Elle fonctionne également en passant des entiers négatifs mais le code correspondant à l'entier n'existant pas, les caractères semblent être associés aléatoirement et ne sont pas dans le bon ordre.

Il ne vaut donc mieux pas de passer d'entiers négatifs en paramètre de PutChar.

PutString:

La fonction PutString a nécessité beaucoup de tests pour savoir si elle fonctionnait correctement. Il a fallu réflechir à propos de la taille de la chaine de caractères passée en paramètre et de la taille de la chaine de caractères maximale que la fonction peut lire en une fois.

Les tests ont donc consisté à modifier et tester à chaque fois les tailles des ces deux éléments: donner des petites et grandes valeurs à MAX_STRING_SIZE et à la chaine de caractères passée en paramètre dans le programme de test.

GetChar:

Pour tester correctement la fonction GetChar, on utilise PutChar avec la valeur de retour de GetChar afin de l'afficher sur la console puis on vérifie si cela correspond bien au caractère que l'on a écrit dans la console au départ.

GetString:

Pour tester correctement GetString, il a fallu beaucoup toucher aux valeurs de MAX_STRING_SIZE, du deuxième paramètre de GetString ainsi qu'à la chaine de caractères entrée par l'utilisateur.

Tout ceci, étant globalement similaire à la méthode utilisée pour PutString, permet de vérifier que l'intégralité de la chaine de caractère entrée par l'utilisateur est bien stockée au fur et à mesure dans le buffer.