

Annexe G

Travaux Pratiques 4

L'objectif de ce TP est de mettre en pratique les concepts présentés dans le chapitre 9 à savoir, la notion d'activité, son exécution et sa mémoire.

Nous vous demandons de télécharger depuis le serveur SVN (voir chapitre 5 le projet TP4_Eleves et de nommer l'importation TP4 (dernier écran d'importation).

Le déroulement du TP est subdivisé en trois parties. La première consiste à valider les acquis du TP2. On vous demande de trouver un bug dans le code fourni en relation avec la gestion des interruptions matérielles. La seconde partie se consacre à la manipulation de la pile. Il s'agit d'afficher puis de modifier l'adresse de retour d'une fonction présente dans la pile. Enfin la dernière partie se focalisera sur la gestion du tas. Nous vous demandons d'implémenter un mini-gestionnaire de mémoire dynamique.

Question "sur le TP précédent" - Durée estimée : 30 minutes

Notre jeu ne fonctionne plus!. En particulier, les raquettes ne montent plus et ne descendent plus lorsque nous appuyons sur les touches qui sont associées à ces mouvements sur le clavier. Pourquoi?

Sous question 1

Modifier le code pour revenir au fonctionnement du précédent TP. Ce code se résume à l'ajout d'une ligne et à la suppression d'une ligne dans la fonction `Sextant_main` du fichier `main.cpp`.

Sous question 2

Une fois le clavier remis en marche, nous vous demandons de compléter la fonction `reboot` dans le fichier `handler_clavier.cpp` se trouvant dans le répertoire :

`sextant/interruptions/handler.`

Cette fonction est appelée lorsque vous taper sur la touche `q`. Nous voulons que notre machine reboot à chaque fois que nous appuyons sur la touche `q`. Pour cela, il suffit d'écrire `0xfe` sur le microcontrôleur du clavier au port `0x64`. Testez.

Pour passer aux questions suivantes mettre en commentaire le code :

Question 3 - Durée estimée 15 minutes

Vous disposez maintenant de l'adresse de retour de `func`. Nous allons chercher dans le code binaire l'instruction associée à cette adresse. Pour cela, désassemblez le code binaire `sextant.elf` en vous aidant d'`objdump` (redirigez la sortie en utilisant "`> resultat.text`"). Pour cela consultez la section 3.6 du chapitre 3 ainsi que de la section 9.2.5 du chapitre 9.

Le code affiché est formaté en colonne. Chacune des lignes contient un certain nombre d'information. Détaillez les informations de la ligne ci-après

```
2 2047ea: e8 91 d4 ff f   call 201c80 <_ZEcran11afficherMotEc7Couleur>
```

Listing G.2 – Source

Que signifie la référence à `<_ZEcran11afficherMotEc7Couleur>`? Comment obtenir une information plus claire? (indice : `demangling`)

Question 4 - Durée estimée 15 minutes

Dans le `main` nous avons l'appel à `func` entouré de deux affectations à la variable `i`. A la fin de l'exécution de `func`, `i` est affecté à 1. Nous allons sauter cette instruction. La première étape consiste à retrouver l'adresse de cette instruction dans le binaire de `sextant.elf`.

```
1 i=0;
2 func(1,2,&ecran);
3 i=1;
```

Listing G.3 – Source

Pour retrouver les instructions assembleur qui affectent `i` à la valeur 1, aidez vous de l'adresse trouvée dans la question 3, en déduire l'adresse de l'instruction après l'affectation de `i` à 1.

Question 5 - Durée estimée 15 minutes

Modifiez la valeur de retour de `func` pour sauter l'instruction "`i=1`". Aidez vous du pointeur `p` de la question 3. Pour cela vous devez inscrire dans `(*p)` l'adresse trouvée à la question précédente.

Question 6 - Durée estimée 30 minutes

Développez un gestionnaire de tas primitif. Pour cela il faut surcharger l'opérateur `new()`, puis complétez un premier gestionnaire situé dans `sextant/memoire/memoire.cpp`

Le principe est le suivant. Un pointeur pointe sur l'adresse du début du tas disponible (initialisé par la méthode `mem_setup`). A chaque demande d'allocation, retournez l'adresse de ce pointeur, puis décaler ce pointeur de `n` octets. Les `n` octet correspondent à la taille mémoire demandée passée en paramètre de la méthode `getmem()`.

Question 7 - Durée estimée 60 minutes

Nous allons implémenter le `delete`. Nous ne vous demandons pas d'implémenter une gestion chaînée des blocs libres, mais simplement, dans un premier temps, de développer une méthode `freemem(int adresse,int taille)` inscrivant des zéros dans les zones mémoires libérées.

Question pour la prochaine fois - Durée estimée 60 minutes

Lorsque vous surchargez l'opérateur `delete`, vous ne récupérez que l'adresse de la zone mémoire à libérer. Trouvez une solution pour retrouver la taille associé à l'objet/`lastrucutre` à désallouer.

Le scénario suivant doit retourner une erreur :

```
1 void main() {  
2   Ecran *ec ;  
  
4   ec= new Ecran ;  
  
6   ec->AfficherMot ("Test1") ;  
7   delete ec ;  
8   ec->AfficherMot ("Test2") ;
```

Listing G.4 – Scénario d'utilisation

BONUS :

- Que se passe t'il avec la pile lors d'un interruption matérielle ?