

Annexe G

Travaux Pratiques 4

L'objectif de ce TP est de mettre en pratique les concepts présentés dans le chapitre 10 à savoir, la notion de multi-activités et les différents types d'ordonnanceurs.

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 TP3. On vous demande de trouver un bug dans le code fourni en relation avec la gestion de la pile d'exécution. La seconde partie se consacre à la manipulation des activités via les méthodes `cpu_context_switch(ctxt *, ctxt *)`, et `create_kernel_thread(char *, (kernel_thread_start_routine_t) *, void *)`. Il s'agit dans cette seconde partie de manipuler deux activités affichant une chaîne de caractère à l'écran. Enfin la dernière partie se focalisera sur la classe Thread.

Sauf pour la dernière partie, l'ensemble du code à écrire ou modifier sera dans le fichier `main.cpp`.

Question 1 "sur le TP n-1" - Durée estimée : 60 minutes

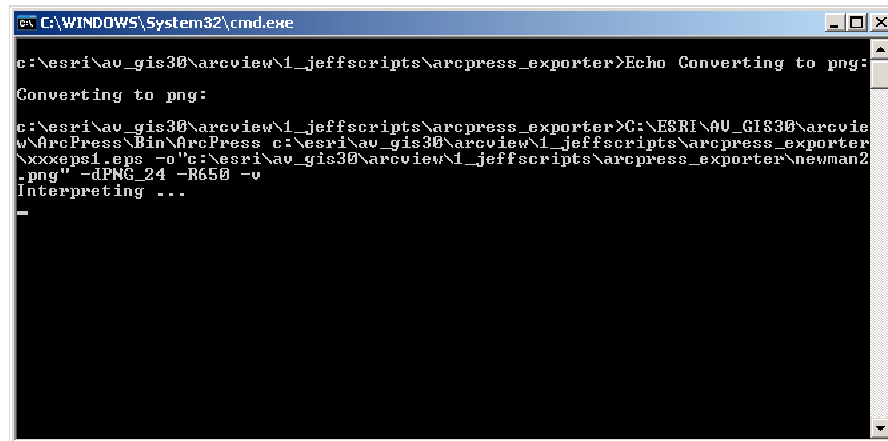
Une faille de type buffer overflow (débordement de tampon) a été insérée dans le code. Une fois retrouvée vous devrez en analyser les causes et comprendre comment elle a été mise en oeuvre. Pour vous aider, le buffer overflow se situe dans la fonction `endlessfunc(Ecran *)`.

Après compréhension du mécanisme, déplacer la déclaration de la variable `tmp` en dessous de `piRet` et diff, compiler et relancer.... que se passe t'il ?

Avant de passer à la question 2, décommentez la ligne 269 (`while(true)`) de la fonction `main`.

Question 2 - Durée estimée : 30 minutes

Nous avons pour cette question développé deux fonctions `Hello1` et `Hello2` qui affichent à l'écran une chaîne de caractère passée en paramètre. En utilisant la fonction `cpu_context_switch` du fichier `sextant\ordonnancements\cpu_context_switch.S`, faites en sorte que le message affiché par les deux hello s'entrelace correctement. C'est-à-dire, que s'affiche à l'écran `Hello World`. Un exemple d'utilisation de la fonction `cpu_context_switch` vous est donné dans la fonction `void print_hello_world ()`.



```

C:\WINDOWS\system32\cmd.exe
c:\nesri\av_gis30\arcview\1_jeffscripts\arcpress_exporter>Echo Converting to png:
Converting to png:
c:\nesri\av_gis30\arcview\1_jeffscripts\arcpress_exporter>C:\ESRI\AV_GIS30\arcvie
w\ArcPress\Bin\ArcPress c:\nesri\av_gis30\arcview\1_jeffscripts\arcpress_exporter
\xxxeps1.eps -o "c:\nesri\av_gis30\arcview\1_jeffscripts\arcpress_exporter\newman2
.png" -dPNG_24 -R650 -v
Interpreting ...

```

FIGURE G.1: Comme back to DOS

Ce système d'ordonnancement coopératif direct était le seul disponible dans le système d'exploitation DOS. Dans une copie d'écran vous est donné en G.1.

Avant de passer à la question 3, commentez la ligne 279 (`while(true)`);

Question 3 - Durée estimée 15 minutes

Etudier les codes du fichier `sextant\ordonnancements\cpu_context.cpp`. A quoi sert la fonction `'core_routine'`. Aidez vous du chapitre 10.

Question 4 - Durée estimée 30 minutes

Dans les premières questions, les activités étaient dans un mode coopératif direct. L'activité associée à `Hello1` faisait directement appel au contexte de l'activité associée à `Hello2`. Un système d'exploitation moderne doit permettre la création dynamique des activités. Le système d'exploitation doit offrir aux activités une fonction leur permettant de relâcher simplement le processeur sans devoir désigner la futur activité à exécuter. C'est l'objectif de la fonction `thread_yield()`. Cette méthode a pour objectif de relâcher le processeur et de passer la main à l'ordonnanceur. Ce dernier choisira une des activités en état prêt. L'ensemble du code de l'ordonnancement coopératif se trouve dans le répertoire éponyme.

Étudiez le code des fonctions `thr_hello1` et `thr_hello2`. Constatez que les threads sont bien créés et coopèrent via l'utilisation de la méthode `thread_yield()`. A part le message de la ligne 297, rien ne s'affiche. Pourquoi?. Solutionnez le problème.

Une des solutions pour résoudre ce problème fait que le message de la ligne 297 ne s'affiche plus. Si vous êtes dans cette situation, expliquez pourquoi?

Vous venez d'implémenter/utiliser un ordonnanceur coopératif tel que celui proposé dans les versions de Windows 1 à 3.11 G.2

Avant de passer à la question 5, commentez la ligne 299 (`while(true)`); et les codes ajoutés dans le `main` pour solutionner le problème.

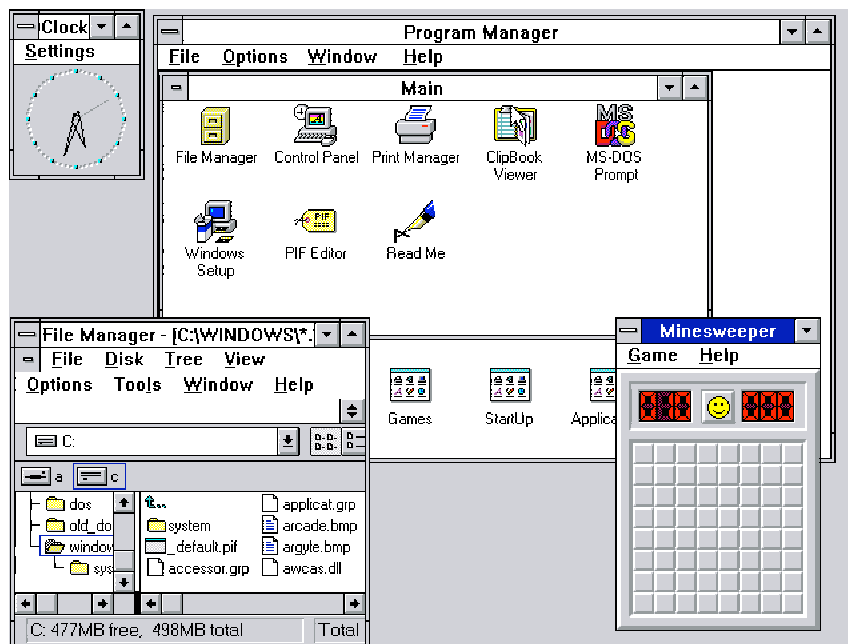


FIGURE G.2: Comme back to Windows 1 to 3.11

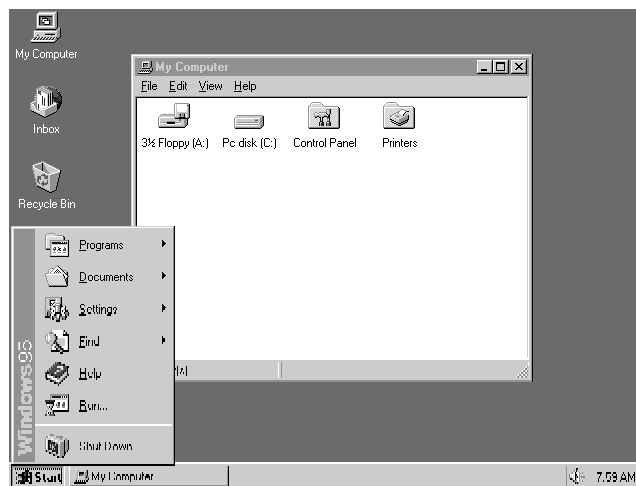


FIGURE G.3: Comme back to Windows 95.

Question 5 - Durée estimée 15 à 45 minutes

Il existe un certain nombre d'inconvénients à utiliser un ordonnaceur coopératif. Entre autres, si une des activités ne souhaite ou ne peut pas relâcher le processeur, les autres activités se retrouvent bloquées dans l'état prêt. Ce problème existait dans les systèmes d'exploitation Mac OS 1 à 9 et Windows 1 à 3.11. Si une application ne marchait plus (plantait) c'est l'ensemble du système qui était bloqué.

Deux solutions sont envisageables. La première consiste à implémenter une fonction **Control+Alt+Del**, la seconde consiste à développer un système permettant de préemp-

ter le processeur à une activité. C'est la seconde solution que nous avons retenue.

Transformez votre code pour accepter l'ordonnancement préemptif. La solution se résume à une seule ligne de code à rajouter dans le `main`. Dans un premier temps nous allons provoquer le changement d'activité en appuyant sur une touche (quelconque du clavier). Dans un second temps, ce changement devra être effectué toutes les `x` millisecondes.

- Indice 1 : chercher la fonction `sched_clock` dans le repertoire preemptif.
- Indice 2 : `IRQ_TIMER` et/ou `IRQ_KEYBOARD`

Après avoir trouver la solution, il devrait s'afficher à l'écran les caractères des quatre chaînes de caractère. L'affichage des caractères devrait s'entrelacer aléatoirement. Ces chaînes sont deux `Howrd` et deux `el ol`

Pourquoi s'entrelacent elles aléatoirement ? Pourquoi 4 chaînes et non deux ?

Bravo, vous venez d'implémenter/utiliser un ordonnanceur préemptif tel que celui implémenté dans windows 95 G.3.

Question pour la prochaine fois - Durée estimée 60 minutes

Implémentez une classe `Thread` dans le repertoire à créer `sextant/Activite`. L'objectif de cette classe est de fournir les mêmes fonctionnalités que la classe Java "Thread". Le listing suivant vous présente les fonctionnalités recherchées.

```

1 class FirstThread extends Thread {
2     int i;
3     FirstThread(int test) { i = test; }
4     public void run() {
5         i=i+1;
6     }
7 }
```

Listing G.1: Main.cpp

```

1 FirstThread p(143);
2 p.start();
```

Listing G.2: Main.cpp

Indices :

```

1 class Threads {
2     ...
3 protected:
4     void Yield(){ thread_yield(); };
5     void Exit(){ ... };
6 public :
7     void start(){ ... };
8     virtual void run(){};
9 };
```

Listing G.3: Main.cpp