Lab Embedded Real-Time Operating System

Systèmes d'exploitation embarqués Temps Réel

Application : (FreeRTOS / Embedded RTLinux)

STUDENT Edition

Ver 3.0-1

Sadok Bazine <sbazine.ens@gmail.com>

Google Classroom EmRTOS-ING3-uElec-25/26

2025/2026

Sommaire

Table des matières

1 Mise en œuvre d'un environnement Linux

Avant propos

L'utilisation d'un espace "virtuel" ou d'un conteneur est une bonne pratique pour isoler vos travaux du reste du système et pour avoir un espace dédié à vos essais pratiques.

Selon le cas, si vous disposez d'un Linux vous pouvez passer directement à la section 1.3. Les utilisateurs d'un Windows doivent suivre les instructions de la section suivante :

1.1 Installation et mise en marche de Ubuntu sous Windows WSL2

Windows 10 /11 disposent des outils WSL (Windows Subsystem for Linux). Pour installer Docker sous WSL2 veuillez suivre les indications suivantes :

1.1.1 Install/update WSL2

- Selon les cas il est parfois nécessaire de :
  - Activer les fonctionnalités Windows :
    - `H-Virt:Hyper Virtualisation`,
    - `WSL2: Soussystème Windows pour Linux`,
    - `Virtualisation sous Windows`...
  - Mettre à jours Windows,
  - Redémarrer ...
- Lancer une fenêtre `PowerShell` en tant qu'administrateur et assurez vous que `WSL2` est à jour :

```
wsl --set-default-version 2
```

```
wsl --update
```

1.1.2 Installer une Distribution Linux sous WSL2

- Installer la distribution Linux par défaut `Ubuntu` :

```
wsl --install
```

- Pour lister les Distributions disponibles :

```
wsl --list --online
```

- Vous pouvez installer une autre distribution par :

```
wsl --install -d <Distro...>
```

1.1.3 Démarrer votre Linux sous WSL2

- Pour lister toutes les distributions installées sur votre Windows :

```
wsl -l -v
```

— Lancer un Shell Linux en faisant une recherche sur votre windows.

— Ou, taper `ubuntu.exe` dans un `PowerShell`.

— Vous pouvez à présent, passer à la section 1.3 d'installation et de mise en marche de Docker ou Linux.

1.2 VM vs Container

Vagrant vs Docker

— *Vagrant* et *Docker* sont parmi les outils utilisés actuellement, pour améliorer l'expérience de développement et augmenter la productivité des ingénieurs ;

— Il se basent sur généralement sur des outils de virtualisation comme `VirtualBox, VmWare, Qemu, Xen...`

— ⇒ Le but de ces outils est la simplification de la phase de développement et de déploiement des applications.

Vagrant vs Docker

Malgré que ces deux outils sont très similaires, nous allons adopter dans ce guide l'outil d'isolation **Docker** :

— Basé sur le concept de **Container** et **Images** ;

— IL est légé et efficace ;

— C'est un outil d'isolation avant d'être un outil de virtualisation ;

— Isoler l'environnement de développement du système qui tourne sur la machine ;

— ⇒ Les installations et les modifications apportées à l'environnement de développement ne vont pas affecter le reste de la machine...

Pour plus de détails sur ces mécanismes, vous pouvez consulter les pages officielles suivantes :

Docker : https://docs.docker.com/get-started/overview/

Vagrant : https://www.vagrantup.com/intro/index

1.3 Installation de Docker

Les instructions suivantes vous permettent d'installer les programmes nécessaires selon la famille de votre système d'exploitation.

1.3.1 Mettre à jour la BD du gestionnaire des packages

Exécuter l'instruction suivante :

`sudo apt update`

1.3.2 Installation de Docker

`sudo apt install docker.io`

ou ...

```
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
↪ docker-compose-plugin
```

1.3.3 Ajouter l'utilisateur au groupe Docker

— Ajouter l'utilisateur que vous avez définie dans votre Shell Linux au groupe Docker par :

`sudo usermod -aG docker $USER`

— Puis redémarrer par :

`sudo reboot`

1.3.4 Tester Docker

Commencer par ouvrir un Shell Linux et exécuter :

```
docker --version
Docker version ...
```

puis vérifier le bon fonctionnement du gestionnaire docker par :

```
docker images
REPOSITORY            TAG       IMAGE ID            CREATED             SIZE
```

1.3.5 Gestion du service système Docker

Selon le besoin, vous pouvez taper les instructions suivantes :

`sudo service docker status` pour afficher l'état du service Docker.

`sudo service docker start` pour démarrer le service Docker au cours de cette section,

`sudo service docker enable` pour activer le démarrage automatique du service docker.

1.4 Mise en œuvre de l'environnement LabCont

Afin d'installer et de mettre en marche l'environnement virtuel, il faut suivre les étapes suivantes :

1. Démarrer un Shell Linux.
2. Afficher le chemin complet de votre répertoire `home`.

   ```
   pwd
   ```

3. Télécharger, dans votre répertoire home `\~`, le fichier `EmSysLabContFiles_Verx.y-z.tar.gz` de la section "**Lab Work Tools**" de votre Google Classroom.
4. Décompresser ce fichier avec :

   ```
   tar -xvzf EmSysLabContFiles_Ver3.2-5.tar.gz
   ```

5. vérifier que vous avez l'arborescence suivante avec la commande `tree`.

   ```
   ~
   ├── EmSysLabContFiles_Ver3.2-5.tar.gz
   ├── LabCont
   │   ├── Buildlabimg
   │   ├── Startlabcont
   │   ├── WorkDir/
   │   ├── container.env
   │   ├── rootsblabimg_3.2-5.tar.gz
   │   ├── stud.Dockerfile
   │   └── student.env
   ```

6. Créer le répertoire `TP-EmRTOS` dans le répertoire `WorkDir`[1]. Une fois cette étape est terminée, vous devez avoir l'arborescence suivante :

   ```
   LabCont/
   LabCont
   ├── ...
   ├── stud.Dockerfile
   └── WorkDir/
       └──TP-EmRTOS/
   ```

7. Éditer le fichier `student.env` pour remplacer le nom d'utilisateur `_2bf_` par le *login* que vous avez reçu par email ou selon le modèle donné par le fichier.

   ```
   USER_NAME=_2bf_
   ```

   Attention, il ne faut pas mettre d'espaces ni des caractères accentués dans votre *login*.

8. Taper les instructions suivantes pour **construire les images de votre conteneur** :

   ```
   ./Buildlabimg
   ```

   Vérifier la présence des images avec :

   ```
   docker images
   ```

   ```
   REPOSITORY          TAG         IMAGE ID            CREATED             SIZE
   sblabimg            3.2-5       xxxxxxxxxxxx        xx hours ago        3.74GB
   rootsblabimg        3.2-5       xxxxxxxxxxxx        xx hours ago        2.31GB
   ```

1. `mkdir -p WorkDir/TP-EmRTOS`

9. Taper les instructions suivantes pour **démarrer le conteneur** `emsyslab` :

   `./Startlabcont`

   Votre fenêtre de travail actuelle doit se transformer en un Shell linux vous donnant accès au fonctionnalités offertes par le *container* `LabCont`. Vous devez avoir une fenêtre similaire à ceci :

```
Starting sb container !!
Starting sblabimg:3.2-5 under the name of emsyslab
mounting ... to /home/sb/WorkDir
sb@emsyslab ~/WorkDir$                                    [21/09/25| 6:41PM]
sb@emsyslab ~/WorkDir$ uname -a                           [21/09/25| 6:41PM]
Linux emsyslab 6.12.45\_1 #1 SMP PREEMPT_DYNAMIC Sat Sep  6 23:06:11 UTC 2025
x86_64 GNU/Linux
```

1.5 Travail à rendre du TP LabCont

Travail à rendre du TP LabCont :

— Si vous allez installer votre `LabCont` pour la première fois, il faut terminer toutes les étapes précédentes avant de procéder à la génération du rapport.

— Si vous avez déjà configuré votre `LabCont` dans une autre matière, référez-vous à la section 1.3.5 pour la gestion du service docker.

Parla suite, vous devez créer les répertoires manquants de l'étape 6 de la section 1.4 puis démarrer votre container selon la section 1.6.1.

— Générer le rapport en suivant les étapes suivantes :

  — Télécharger le script `GenLWRep-LabCont.sh` dans le répertoire `TP-EmRTOS`, puis l'exécuter dans le même répertoire :

  `sh GenLWRep-LabCont.sh`

  — Il faut rendre les fichiers séparément sur la plateforme *Google Classroom*.

  — **Le fichier à rendre :** `<VotreLogin>-REPORT--LabCont-TP-EmRTOS.log`.

1.6 Utilisation de l'environnement LabCont

1.6.1 Mettre en marche l'environnement LabCont

Pour mettre en marche votre conteneur il faut :

1. Se mettre dans le répertoire `LabCont`,
2. taper : `./Startlabcont`

Ce qui ouvrira un terminal interactif avec le système dont on a besoin. Pour sortir de ce terminal il suffit de taper `exit`.

1.6.2 Espace partagé entre votre système et l'environnement LabCont

Le répertoire `WorkDir` est un espace de travail commun entre votre système hôte et le conteneur, toute modification apportée à ce répertoire est visible par les deux systèmes.

2 Introduction au Shell Linux

2.1 Quelques notions du Shell

Quelques notions du Shell

Manipulation des répertoires

— Sous Linux, la ligne de commande est un outil très performant. Très souvent, c'est le seul langage de script disponible sur la cible. Ces travaux ont pour objectif de vous familiariser avec cet environnement et d'explorer le potentiel d'un shell Linux.

— Démarrer l'environnement virtuel (voir section 1.6.1) ou vos cartes *Raspberry* et connectez-vous à vos comptes respectifs qui vous sont fournis par email ou lors de la première séance.

Manipulation des répertoires

— Entrer dans le répertoire `TP-EmRTOS` [2], créer le répertoire `Shell` et placez-vous dedans [3] ,

`mkdir Shell`

— Entrer dans ce répertoire :

`cd Shell`

— Création des répertoires :

```
mkdir dir1 dir2 dir3
mkdir -p dir1/dir11/dir111
mkdir -p dir1/dir12/dir121
mkdir dir1/dir12/dir122
tree -L 1
tree -L 2
```

Conseils :

— Consulter l'`Aide-mémoire des commandes Shell` (Format A3 *Google Classroom*) dont vous disposez, pour se rappeler des commandes Shell ainsi que de leurs syntaxes.

— N'hésiter pas à consulter l'aide des commandes par l'option `--help` ou la documentation système par la commande `man` puis la commande à consulter.

— Fonctions utiles : `mkdir, rm, ls, tree, pwd, mv, nano, touch, wc, ...`

— `[TAB :]` permet d'aller plus vite et se protéger des fautes de frappe

```
cd /h<TAB>/j<TAB>/c<TAB>
```

2.2 Redirections des entrées/sorties

Redirections

On utilise beaucoup les redirections des entrées/sorties sous Linux (`< |,<< et > >>`) :

2. `cd && cd WorkDir/TP-EmRTOS`
3. `mkdir Shell && cd Shell`

— Sortie standard vers un fichier

```
echo "***TP-SHELL***"  > dir1/file1
ls -l > dir1/file1
cat dir1/file1

echo "***TP-SHELL***"  > dir1/file2
ls -l >> dir1/file2
cat dir1/file2

tree >> dir1/file2
cat -n dir1/file2 > dir1/file21

cat dir1/file21

cat -n dir1/file2 dir1/file21

touch dir2/file2
touch dir2/file22 dir2/file222
tree
tree >> dir2/file222

ll dir*/file? > dir2/file2
ll dir*/file?? > dir2/file22
ll dir*/file*
```

— Redirection vers l'entrée standard

```
cat -n
hello
From TP-Shell
<Ctl+d>
cat -n > dir2/file3
hello
From TP-Shell
<Ctl+d>
```

<Ctl+d> : Appuyer simultanément sur les touches `Ctrl` et la touche `d` du clavier pour interrompre la commande `cat -n`.

```
cat -n < dir1/file2
```

Redirections II

— Sortie standard d'une commande vers l'entrée d'une autre

```
cd dir1
echo Hello From Linux shell | wc
ls | wc -l
ls | cat -n
```

— Couplage des redirections

```
cat -n < file2 | wc > file3
```

— L'espace n'est pas obligatoire et les redirections ne sont pas forcement à la fin de la ligne

```
>file3 cat<file2 -n
```

Les chemins

Il est possible d'utiliser des chemins :

— absolus

```
cd && pwd
```

```
cat /home/<VotreLogin>/WorkDir/TP-EmRTOS/Shell/dir1/file3
```

— relatifs

```
cd ~/WorkDir/TP-EmRTOS/Shell/dir2
```

```
pwd
```

```
cp ../dir1/file? ./
```

```
tree
```

```
tree ..
```

Quelques notions de shell

Globbing :

```
rm *.o
```

Alias :

```
alias msg="echo \"***TP-SHELL***\""
alias tree1="tree -L 1"
alias tree2="tree -L 2"
msg && tree1 && tree2
```

2.3 Script shell

Écrire un script shell

Les fonctionnalités d'un script shell sont identiques à celles de la ligne de commande.

— Télécharger (et renommer) le(s) fichier(s) `2bf.script.sh`, dans le répertoire `Shell`. Vous devez avoir dans votre répertoire `Shell` la liste suivante :

```
├── Shell
   ├── dir1
   │   └── ...
   ├── ...
   ├── script.sh
```

On suffixe généralement les scripts shell par .sh. Éditer le fichier par

`nano script.sh`

— La première ligne,`##!/bin/sh` indique le shell utilisé.

— La suite s'écrit comme sur la ligne de commande

— Enregistrer <Ctrl+O> puis <Enter>, fermer par <Ctrl+X>.

— Exécution du script

```
sh script.sh
```

— Exécution autonome du script ; Il est nécessaire de lui donner les droits en exécution :

```
chmod +x script.sh
./script.sh
```

— Ajouter des commentaires explicatifs pour chaque ligne du fichier `script.sh`. Sachant que les commentaires commencent par le caractère "`##`".

2.4 Manipulation des Expressions régulières

Création de l'espace de travail

— Nous supposons, dans tous ce qui suit que vous êtes dans le répertoire `Shell`.

Expressions régulières

— Télécharger (et renommer) le(s) fichier(s) `2bf.RegExp.sh`, dans le répertoire `Shell`. Vous devez avoir dans votre répertoire `Shell` la liste suivante :

```
├── Shell
    ├── dir1
    │   └── ...
    ├── ...
    ├── RegExp.sh
```

— Analyser le contenue du fichier `RegExp.sh`, puis utiliser `ifconfig` ou `ip a` et `grep` pour extraire l'adresse IP de ta machine.

— faites le même travaille avec la commande `sed`

— ajouter des commentaires explicatifs au fichier `RegExp.sh`.

— Lancer un terminal `bash` avec la commande :

```
bash
```

— créer votre propre commande `MyIP` dans le fichier .bashrc :

```
alias myip="..."
```

— tester le fonctionnement de cette commande.

— quitter le terminal `bash` avec la commande `exit` pour poursuivre le travail sur le terminal par défaut.

2.5 Travail à rendre du `TP Shell`

Travail à rendre du `TP Shell` :

— Une fois vous avez terminé toutes les étapes précédentes ;

— Télécharger le script `GenLWRep-Shell.sh` dans le répertoire `Shell`, puis l'exécuter dans le même répertoire :

`sh GenLWRep-Shell.sh`

— Il faut rendre les fichiers séparément sur la plateforme *Google Classroom*.

— **Les fichiers à rendre :** `<VotreLogin>-REPORT--Shell-TP-EmRTOS.log`, et fboxscript.sh `RegExp.sh`.

3 Gestion des processus sous Linux

Introduction : Exécution dans un environnement multicores

— La gestion de la mémoire pour l'exécution des processus est un peu complexe sous Linux. Mais l'utilisateur n'a pas à se soucier de ces détails. Les couches d'abstraction d'allocation mémoire, de changement de contexte, de gestion de la mémoire virtuelle ... font très bien ce travail.

— Le processeur arrive à exécuter plusieurs processus avec l'aide de l'ordonnanceur. Ce dernier veille sur la répartition des ressources CPU sur tous les processus en cours d'exécution. Ainsi, le CPU est alloué à plusieurs processus successivement et très rapidement, ce qui donne à l'utilisateur l'illusion du multitâches (parallélisme).

— Même avec les processeurs multicores, ce mécanisme de gestion des processus reste vrai. En fait, avec un nombre de processus supérieur à celui des cores, l'ordonnanceur doit aussi intervenir pour fixer la politique d'allocation de ces ressources.

— Afin d'utiliser des threads sous Linux, il faut compiler le programme `C` avec l'option `-lpthread` :
`gcc -o <nomexecutable> -lpthread nomprog.c`

— Actuellement la majorité des systèmes à microprocesseur disposent d'une architecture multi-cores. Afin de limiter l'exécution d'un *process* sur un seul core, vous pouvez l'exécuter avec la commande :

`## taskset 01 ./process` [4]

Pour avoir une idée sur le nombre de cores dont un process, d'identifiant `<pid>`, peut utiliser, il suffit de taper :

`## taskset -p <pid>`

Création de l'espace du travail

— Démarrer l'environnement virtuel (voir section 1.6.1) ou vos cartes *Raspberry* et connectez-vous à vos comptes respectifs qui vous sont fournis par email ou lors de la première séance.

— Entrer dans le répertoire `TP-EmRTOS` [5], créer le répertoire `Process` et placez-vous dedans [6] afin d'avoir l'arborescence suivante :

```
├── TP-EmRTOS
    ├── Shell
    └── Process
```

— Nous supposons, dans tout ce qui suit que vous êtes dans le répertoire `TP-EmRTOS/Process`.

4. installation : sudo apt-get install util-linux
5. `cd && cd WorkDir/TP-EmRTOS`
6. `mkdir Process && cd Process`

3.1 La commande `fork()` et clonage de processus

Récupération des fichiers modèles

— Entrer dans le répertoire `TP-EmRTOS/Process`

— Télécharger (et renommer) le(s) fichier(s) `2bf.process.c`, `2bf.status.c`, `makefile`, `PrintPIDStatus.c` et `libPIDStatus.h` dans le répertoire `Process`.

Vous devez avoir dans votre répertoire `Process` la liste suivante :

```
├── Process
    ├── makefile
    ├── libPIDStatus.h
    ├── PrintPIDStatus.c
    ├── process.c
    └── status.c
```

— le fichier `makefile` vous permet d'automatiser la tâche de compilation :

`make process` pour compiler le fichier `process.c`,

`make status` pour compiler le fichier `status.c`

et `make PrintPIDStatus` pour compiler le fichier `PrintPIDStatus.c`

Vous pouvez aussi utiliser les options de nettoyage `make clean` et `make mrproper` pour supprimer les fichiers de compilation.

La commande `fork()` et clonage de processus

1. Le programme `process` donne un exemple type d'exploitation du potentiel de la commande `fork()` sous Linux. Compléter ce programme pour qu'il ait un comportement similaire à ce qui suit.
2. Compiler puis exécuter le programme par :

   ```
   make process && ./process
   ```

   pour avoir un affichage qui ressemble à ce qui suit :

   ```
   Main : Mon pid est 100
   Père : Mon pid est 100;
       le pid de mon père est 1;
       Le pid de mon fils est 101.
   Père
   Fils : Mon pid est 101;
       le pid de mon père est 100.
       Fils
   Père
       Fils
   Père
   ```

   — Compiler et analyser le comportement du programme en variant les capacités respectives du père et du fils : `Cp` et `Cf`

— Analyser les taux d'utilisation du CPU, à partir d'un autre terminal Linux ou d'un autre CMD Windows en démarrant un nouveau Shell `bash` sur votre *container* via la commande :

`docker exec -it <container id> bash`

pour identifier votre *container* taper la commande :

`docker container ls`

— Recompiler et exécuter le programme en augmentant la charge de calcul de ce dernier. Suspendre l´exécution du programme avec la commande `^Z` puis utiliser les commandes `ps, pstree` et `htop` pour afficher superviser l'exécution des *process* en cours.

Remarque : vous pouvez utiliser les instructions `fg` et `bg` pour respectivement, reprendre un programme suspendu ou suspendre un programme en cours d'exécution.

— Ré-exécuter et suspendre le programme de nouveau et analyser l'état de ce dernier par les instructions suivantes.

`cat /proc/<pid>/status | grep State`

— Compiler le programme `PrintPIDStatus.c` puis analyser l'état du processus de la question précédente par ce programme :

3. Ajouter les lignes suivantes à la fin de la fonction `process_pere()`.

```
1  printf("Père : Synchronization  sur la fin du processus fils \n");
2  wpid = waitpid(pidFils,&status,0);
3  printf("Père : Fin du processus fils de pid %d\n", wpid);
```

pour avoir un affichage qui ressemble à ce qui suit :

```
...
    Fils
Père
    Fils
Père : Synchronization  sur la fin du processus fils
    Fils
    Fils
Père : Fin du processus fils de pid 101
```

3.2 Consultation de l'état d'un processus

États d'un processus

On veut faire passer un processus, et son fils, par tous les états de la figure 1, en examinant le contenu du fichier spécial `/proc/<pid>/status`, où `pid` est l'identifiant du processus observé :

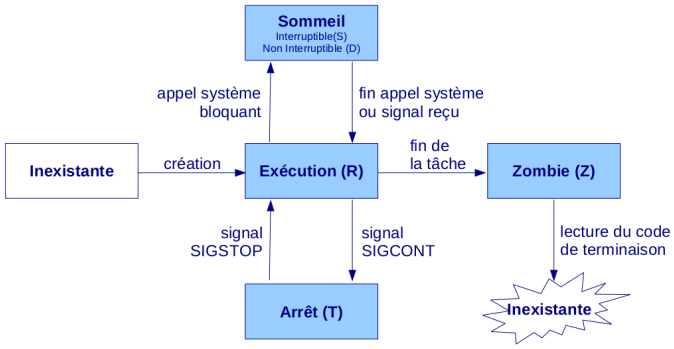

FIGURE 1 – Les états d'un processus Linux

3.2.1 Scénario à implémenter

Scénario à implémenter

1. le processus père va consulter son propre état dans le système de fichier `/proc` puis va se cloner à l'aide de la primitive `fork` avant de s'endormir pendant 5 secondes,
2. le processus fils va profiter du sommeil du père pour examiner l'état de celui-ci et l'afficher (en accédant encore au système de fichier `/proc`). Il se terminera immédiatement après.
3. À son réveil, le processus père examinera et affichera l'état du processus fils, avant et après avoir lu le `status` de retour (appel à `wait` ou `waitpid`) .

3.2.2 Démarche à suivre

Démarche à suivre

Les étapes suivantes nous permettent de suivre l'état du process en utilisant le système de fichier `/proc` :

1. La fonction `void PrintPIDStatus(pid_t)` de la bibliothèque `libPIDStatus.h` prend en paramètre le `pid` du *process* à examiner et affiche l'état observé dans le fichier système `/proc/<pid>/status`

   Compléter le programme `status.c` afin d'implémenter le scénario décrit dans la section 3.2.1.
2. Compiler le programme : `make status`
3. Lancer le programme à partir d'un premier terminal, prenant comme exemple `pid=53` :

```
./status
MAIN: PID(316)  :  ==>Process(316) : R (running)
Père : sleeping !!

     Fils : Etat du père :  ==>Process(316) : S (sleeping)
     Fils : Je me termine.

Père : Etat du fils avant wait(...)  :  ==>Process(317) : Z (zombie)
Père : Etat du fils après wait(...) :  ==>Process(317) Does not exist !
Père : Je me termine.
```

3.3 Travail à rendre du TP Process

Travail à rendre du TP Process :

— Une fois vous avez terminé toutes les étapes précédentes ;

— Télécharger le script `GenLWRep-Process.sh` dans le répertoire `Process`, puis l'exécuter dans le même répertoire :

```
sh GenLWRep-Process.sh
```

— Il faut rendre les fichiers séparément sur la plateforme *Google Classroom*.

— **Les fichiers à rendre :** `<VotreLogin>-REPORT--Process-TP-EmRTOS.log`, `process.c` et `status.c`.

4 Manipulation des threads sous Linux

4.1 Création des threads

Création de l'espace du travail

— Démarrer l'environnement virtuel (voir section 1.6.1) ou vos cartes *Raspberry* et connectez-vous à vos comptes respectifs qui vous sont fournis par email ou lors de la première séance.

— Entrer dans le répertoire `TP-EmRTOS`[7], créer le répertoire `Threads` et placez-vous dedans[8] afin d'avoir l'arborescence suivante :

```
├── TP-EmRTOS
    ├── Shell
    ├── Threads
    └── Process
```

— Nous supposons, dans tout ce qui suit que vous êtes dans le répertoire `TP-EmRTOS/Threads`.

Récupération des fichiers modèles

— Entrer dans le répertoire `TP-EmRTOS/Threads`

— Télécharger (et renommer) le(s) fichier(s) `2bf.threads.c`, `libThAttr.h`, et `makefile` dans le répertoire `Threads`.

Vous devez avoir dans votre répertoire `Threads` la liste suivante :

```
├── Threads
    ├── makefile
    ├── libThAttr.h
    └── threads.c
```

— La bibliothèque `libThAttr.h` vous propose la fonction

```
display_pthread_attr(pthread_attr_t, char *)
```

permettant d´afficher les informations d´une structure de type `pthread_attr_t`, en préfixant l´affichage des champs de la structure par le paramètre `char*`.

— Le fichier `makefile` vous permet d'automatiser la compilation du fichier `threads.c` :

```
make threads
```

Vous pouvez aussi utiliser les options de nettoyage pour supprimer les fichiers de compilation indésirables.

Création des Threads

— Ce programme donne un exemple type d'exploitation du potentiel des threads sous Linux.

7. `cd && cd WorkDir/TP-EmRTOS`
8. `mkdir Threads && cd Threads`

— Ajouter et/ou commenter les lignes nécessaires[9] pour avoir un comportement similaire à ce qui suit.

```
./threads
===========
Default-Attr :Detach state        = PTHREAD_CREATE_JOINABLE
Default-Attr :Scope               = PTHREAD_SCOPE_SYSTEM
Default-Attr :Inherit scheduler   = PTHREAD_INHERIT_SCHED
Default-Attr :Scheduling policy   = SCHED_OTHER
Default-Attr :Scheduling priority = 0
Default-Attr :Guard size          = 4096 bytes
Default-Attr :Stack address       = (nil)
Default-Attr :Stack size          = 0x0 bytes
===========

je suis le thread N°1.                  je suis le thread N:7.
    Mon pid est 299.                        Mon pid est 299.
    Mon tid est 139796114851584             Mon tid est 139796064495360
    1-Thread N°1                            1-Thread N°7
je suis le thread N°2.                      2-Thread N°9
    Mon pid est 299.                        2-Thread N°6
    Mon tid est 139796106458880             2-Thread N°5
    1-Thread N°2                            2-Thread N°3
je suis le thread N°3.                  ...
    Mon pid est 299.                        9-MAIN process
    Mon tid est 139796098066176             10-Thread N°9
    1-Thread N°3                            10-Thread N°7
...                                         9-Thread N°8
MAIN process: PID= 299, TID= 139796114855744-Thread N°10
    1-MAIN process                      ...
```

— Modifier le nombre des *threads* `THN`, le nombre des itérations `MSGN` et analyser le comportement de ces tâches[10]

4.2 Synchronisation des threads

Synchronisation des threads

— Décommenter et compléter les lignes qui permettent de synchroniser le fonctionnement de ces threads pour avoir un comportement similaire à ce qui suit :

```
./threads
===========
Default Attr :Detach state        = PTHREAD_CREATE_JOINABLE
Default Attr :Scope               = PTHREAD_SCOPE_SYSTEM
Default Attr :Inherit scheduler   = PTHREAD_INHERIT_SCHED
Default Attr :Scheduling policy   = SCHED_OTHER
Default Attr :Scheduling priority = 0
Default Attr :Guard size          = 4096 bytes
Default Attr :Stack address       = (nil)
Default Attr :Stack size          = 0x0 bytes
===========
je suis le thread N°1.
     Mon pid est 362.
     Mon tid est 139818627380992
     1-Thread N°1
je suis le thread N°3.
...
MAIN : Mon pid est 362, mon tid est 139818627385152.
MAIN : J'ai cree les threads de tid [139818627380992 ..139818551846656]
     1-MAIN Process
je suis le thread N°5.
```

9. Les lignes avec `pthread_join`.
10. avec les commandes `ps, pstree, htop ...`

```
     Mon pid est 362.
...
     2-Thread N°2
     2-MAIN Process
     2-Thread N°1
     2-Thread N°7
...
     9-Thread N°3
     10-Thread N°8
     10-Thread N°9
...
MAIN : synchronisation sur la fin du thread N°2 (tid 139818618988288)
     ==> status : 102
MAIN : synchronisation sur la fin du thread N°3 (tid 139818610595584)
     ==> status : 103
MAIN : synchronisation sur la fin du thread N°4 (tid 139818602202880)
     ==> status : 104
...
MAIN : synchronisation sur la fin du thread N°10 (tid 139818593810176)
     ==> status : 110
```

— Modifier le programme `threads.c` de façon à rendre le thread 5 `DETACHED` en appelant la fonction `pthread_detach(...)` :[11]

— Récupérer, avec la fonction `pthread_getattr_np(ThList[4], &Tattr)`, les nouveaux attribues du thread 5 et afficher les avec la fonction `display_pthread_attr`. Vérifier que la modification a bien été apportée au thread en question.

```
...
============
Thr N°5 Attr :Detach state        = PTHREAD_CREATE_DETACHED
Thr N°5 Attr :Scope               = PTHREAD_SCOPE_SYSTEM
Thr N°5 Attr :Inherit scheduler   = PTHREAD_INHERIT_SCHED
Thr N°5 Attr :Scheduling policy   = SCHED_OTHER
Thr N°5 Attr :Scheduling priority = 0
Thr N°5 Attr :Guard size          = 4096 bytes
Thr N°5 Attr :Stack address       = 0x7f2a0d1e0000
Thr N°5 Attr :Stack size          = 0x800000 bytes
============
MAIN : Mon pid est 362, mon tid est 139818627385152.
MAIN : J'ai cree les threads de tid [139818627380992 ..139818551846656]
     1-MAIN Process
je suis le thread N°5.
     Mon pid est 362.
     Mon tid est 139818593810176
     1-Thread N°5
je suis le thread N°7.
     Mon pid est 362.
...
     2-Thread N°7
     2-Thread N°5
     3-Thread N°1
...
     10-Thread N°8
     10-Thread N°1
     9-Thread N°10
...
MAIN : synchronisation sur la fin du thread N°1 (tid 139818627380992)
     9-Thread N°2
     10-Thread N°4
...
MAIN : synchronisation sur la fin du thread N°5 (tid 139818593810176)
     !! pthread\_join error <== Thread N°5 !!
MAIN : synchronisation sur la fin du thread N°6 (tid 139818585417472)
     ==> status : 106
...
MAIN : synchronisation sur la fin du thread N°10 (tid 139818560239360)
```

11. Vous pouvez modifier cet attribue à la création des threads en modifiant la structure `Tattr` avec l'instruction : `err = pthread_attr_setdetachstate()(&Tattr,PTHREAD_CREATE_DETACHED);`

```
==> status : 110
```

— Re-exécuter ce programme plusieurs fois et expliquer l'apparition du message d'erreur :

`segmentation fault ./threads`

4.3 Travail à rendre du TP Threads

Travail à rendre du TP Threads :

— Une fois vous avez terminé toutes les étapes précédentes ;

— Télécharger le script `GenLWRep-Threads.sh` dans le répertoire `Threads`, puis l'exécuter dans le même répertoire :

```
sh GenLWRep-Threads.sh
```

— Il faut rendre les fichiers séparément sur la plateforme *Google Classroom*.

— **Les fichiers à rendre :** `<VotreLogin>-REPORT--Threads-TP-EmRTOS.log` et `threads.c`.