UNIVERSITÉ DE
SHERBROOKE

TP1 – Rapport Collectif

IFT585

https://depot.dinf.usherbrooke.ca/dinf/cours/h20/ift585/tp1---groupe---h.git

Olivier Perrault – 16212377
Cédrick Lemay – 17019575
Jessica Gosselin - 16038208

Explication du code

tp1.cs

- Program.Main(string[] args)
    - Point d'entrée du programme, nécessite la destination du fichier de paramètres en entrées.
    - Assigne le rôle de chaque machine comme machine émettrice ou machine réceptrice
    - Créent 6 threads et commence chacune d'elles

Parameters.cs

- Parameters.TryDeserialise(string file, out Parameters parameters)
    - Permets de désérialiser le fichier de paramètres afin d'obtenir la structure de données « Parameters » contenant les paramètres du programme

Machine.cs

- Regroupe les 3 niveaux de couches à l'intérieur d'un même objet.
- Résous les dépendances entre les différentes couches ainsi qu'avec le support de transmission

Layer.cs

- Contiens les classes abstraites Layer et MachineLayer qui encapsulent un thread et une mémoire synchronisés.
- Encapsule aussi un EventStream afin de pouvoir gérer l'événement le plus approprié
- MachineLayer
    - UpTransferBuffer
        - Mémoire synchronise permettant d'envoyer de l'information a la couche du dessus sans perdre ou sauter d'information
    - DownTransferBuffer
        - Même chose que la UpTransferBuffer pour transmettre à la couche du dessous.

SyncedBuffer.cs

- Contient la classe abstraite « SyncedBuffer » permettant de facilement résoudre le problème de producteur/consommateur lorsque les diffèrent thread accède à une mémoire partagé afin de pouvoir transférer de l'information d'une couche a une autre.
- SimpleSyncedBuffer
    - Classe concrète dérivant «SyncedBuffer » permettant de synchroniser l'accès mémoire à un tableau d'octets fixe.

A3_UserDataLayer.cs

- Contiens le point de lecture du fichier lorsqu'il s'agit d'une machine émettrice et le point d'écriture lorsqu'il s'agit d'une machine réceptrice.
- En mode émetteur, créer une structure « UserData » à partir des octets lus depuis le fichier source.
- En mode récepteur, écris dans un fichier le contenu de « UserData » reçu

- UserData
    - Size
        - Nombre total d'octets requis pour le format de données à cette couche
    - HeaderSize
        - Nombre d'octets requis pour les métadonnées.
    - IsEOF
        - Indique si cette donne utilisateur représente une fin de fichier.
    - OccupiedSize
        - Nombre d'octets réellement occupés par des donnes de fichier
        - Utile afin de représenter fin de fichier.
    - Content
        - Contenu de fichier

A2_Timer.cs

- Gère les « Timer » propres à la partie « Sender » de l'algorithme « Selective Repeat ARQ »

A2_FrameDataManipulationLayer.cs

- En mode émetteur, accumule plusieurs « UserData » jusqu'à obtenir assez d'information pour construire une structure « Frame » et relai ensuite à la couche du dessous.
    - Gère l'algorithme de rejet sélectif à l'aide de « A2_SenderWindow »
        - Si espace libre dans la fenêtre, on stocke le « Frame » construit et commence le timer
        - Si aucun ACK reçu au moment où le « Frame » fait un « Timeout » renvoie la « Frame »
        - Si obtiens un ACK, on déplace la fenêtre jusqu'à l'élément indique par l'ACK
- En mode récepteur, sépare la structure « Frame » reçue en plusieurs structures « UserData » avant de relayer l'information à la couche du dessus.
    - Gère l'algorithme de rejet sélectif à l'aide de « B2_ReceiverWindow »
        - Cas 1 : Obtiens le « Frame » dans l'ordre voulu, simplement déplacer la fenêtre
        - Case 2 : Obtiens le « Frame » dans la fenêtre, mais pas dans l'ordre
            - Conséquence : Envoyer NAK indiquant prochain dans l'ordre
- Case 3 : Between échoué, on ignore le « Frame »
- Frame
    - ID
        - Identifie la trame uniquement pour une exécution du programme
    - Type (Data, EndOfTransmission)
    - OccupiedSize
    - Content

A1_FrameSupport.cs

- En mode émetteur, créer un format conforme à la spécification de « FrameSupport » et relais l'information vers « TransmissionSupport »

    - Se charge d'encoder la trame à l'aide de l'algorithme de «Hamming »
- En mode récepteur, récupère l'information de « TransmissionSupport », extraite les données de la trame et relais l'information a la couche supérieure.
    - Se charge de décoder la trame à l'aide de « Hammin »
    - Corrige une erreur simple d'un bit en mode correcteur
    - En mode détecteur si erreur détectée la trame est perdue.

A1_Hamming.cs

- Contiens le code pour encoder avec Hamming
- Contiens le code pour décoder avec Hamming
- Contiens le code pour détecter avec Hamming

C_TransmissionSupport.cs

- Reçois l'information dans le format « TransmissionSupport » d'une machine et stocke l'information dans un buffer circulaire jusqu'autre machine, soit prête, à recevoir.
- Affiche les erreurs introduites selon le mode sélectionné dans les paramètres.
- Simule le délai du support de transmission selon la valeur des paramètres

Configurations

Plusieurs paramètres ont été ajoutés afin de faciliter le débogage (print). Vous pouvez vous référer au fichier ‘parametres-debug-template.txt’

- Les options de débogage sont désactivées par défaut.
- Pour chaque paramètre additionnel
  - La chaine de caractère contenant seulement « - » désactive le prochain paramètre
  - N’importe quelle autre chaine de caractères sur la ligne active le prochain paramètre

e.g parametres.txt (seulement timeout, erreur, erreur détectée)

```
Resources/pic.png
Output/output.png
1, 5, 65
C
234,445,290
100
0
85
36
100
100
R
B
-
-
-
a2 - timeout
-
-
-
c - error
-
-
-
b1 - detected
-
-
-
-
```

Procédure d’exécution

1. Configurer le fichier de paramètre relativement au dossier où exécute du programme
2. Compiler le programme a l’aide du compilateur mono
   a. csc *.cs
3. Exécuter le programme a l’aide de mono
   a. Spécifier le fichier paramètre relativement au dossier où exécute du programme.
   b. mono tp1.exe parametres.txt
4. Terminer l’exécution à l’aide de control-c

Analyse et conception

La première étape était de parvenir à encapsuler a logique d'une machine à l'intérieur d'une classe. Par la suite, il fut avantageux de faire de même avec les différents threads afin de plus facilement réutiliser la logique. L'utilisation de « SyncedBuffer » représente un point de divergence avec les spécifications du devoir. Le « SyncedBuffer » me permet de parvenir à une approche plus efficace qu'avec l'utilisation des booléens « ReadyToReceiveFromUp », « ReadyToReceiveFromDown », « ReadyToSendUp » et « ReadyToSendDown » qui auraient requis de l'attente active. A la place, « SyncedBuffer » est muni de deux sémaphores « Empty_ReadyToReceive » et « Full_ReadyToSend » verrouillant le thread lorsque la mémoire et vide et pleine respectivement. Puisque chaque thread est muni de deux « SyncedBuffer », le programme possède le même comportement que le programme spécifié.

Ce qui fonctionne

Je crois que tout fonctionne sauf s'il y aurait eu une incompréhension de l'énoncé. Puisque nous n'avons pas eu la correction du premier devoir SVP soyez compréhensible. Merci