

Projet de Programmation Unix

L3 Informatique

2019/20

Michel Soto



SOMMAIRE

Objectif, Organisation et Évaluation	3
Présentation	4
Description détaillée	5
Cahier des charges	7
Travail demandé	9
Annexe	11

1. Objectif, Organisation et Évaluation

Objectif

Mise en œuvre et maîtrise des processus, signaux et fichiers.

Nombre d'étudiants par groupe

Le projet est réalisé par groupe de deux étudiants.

Constitution des groupes

Les étudiants forment les groupes à leur convenance.

Déclaration des groupes

Un étudiant du groupe envoie, **au plus tard le 22/11/2019 16H00**, un mail à :

michel.soto@parisdescartes.fr

ayant pour sujet : [PU] GROUPE PROJET

et dans le corps la liste des membres du groupe avec pour chacun N°ETUDIANT NOM Prénom

Intégration dans la note finale

Le projet n'est pas obligatoire. Il est noté sur 20 et sa note s'ajoutera, sur la forme de 0 à 3 points de bonus, à la note de contrôle continu.

Les étudiants d'un même groupe peuvent avoir des notes différentes.

La note finale à l'**UE** de chaque étudiant sera :

$$\text{MAX} [((\text{Bonus_projet} + \text{DST}) + \text{examen})/2, \text{examen}]$$

Évaluation

L'évaluation sera faite à partir des livrables et d'une mini soutenance individuelle.

Les professionnels de l'informatique doivent être capables de respecter de manière stricte un cahier des charges.

Dans le cadre de ce projet, le non-respect du cahier des charges sera pénalisant car:

1. en tant que futur professionnel de l'informatique, vous devez vous exercer à respecter un cahier des charges
2. le non-respect du cahier des charges rendra votre travail difficile à évaluer

Ces mêmes professionnels utilisent les bonnes pratiques de programmation dans les codes qu'ils écrivent (commentaires, indentation, noms de variables parlants). Vous devez les utiliser aussi.

Dates

06/09/2017: Début du projet.

04/12/2019: Livrable 1 : Compréhension du problème

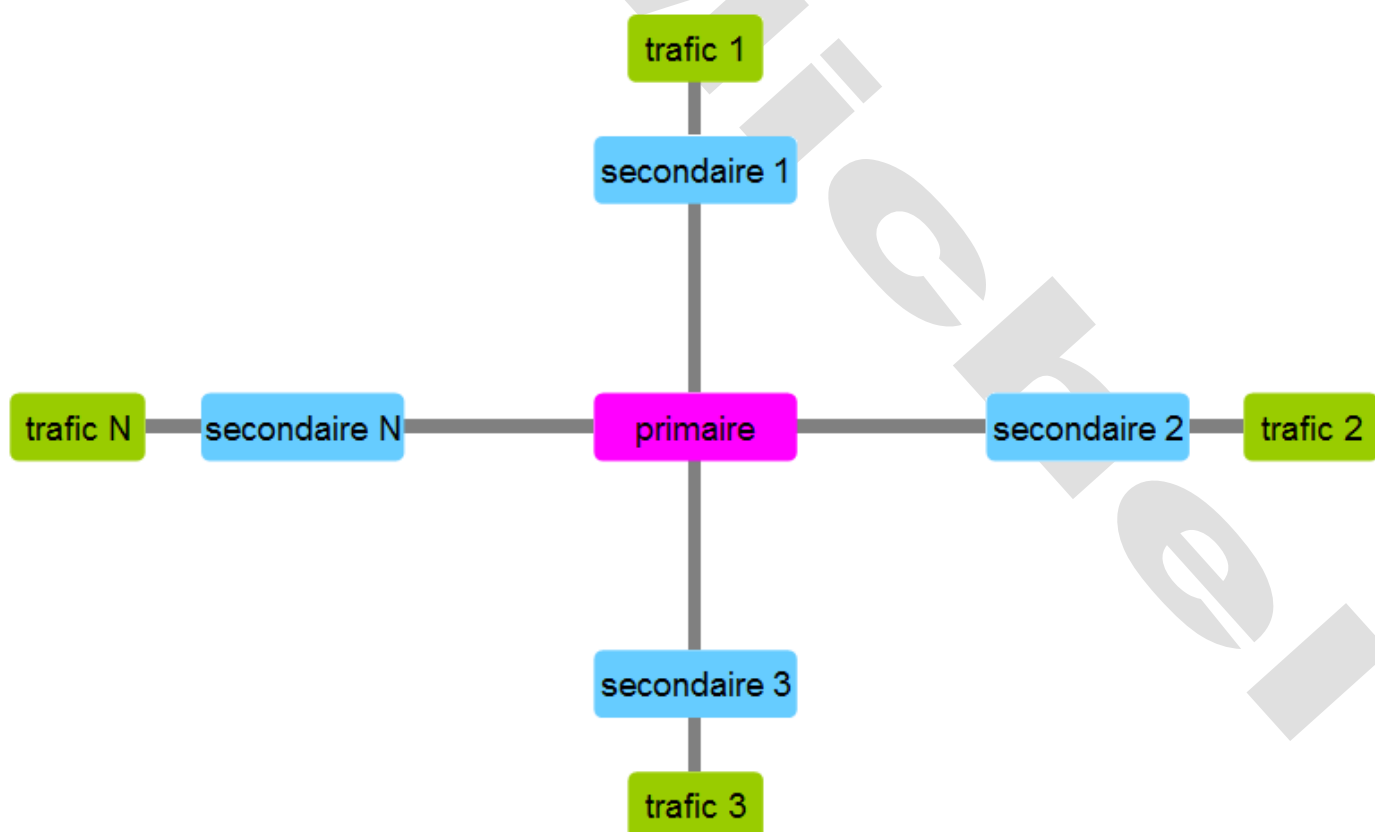
12/12/2019: Livrable 2 : code de `poll_config 1.0`

15/01/2020:

- **Livrable 3 :** code de `poll_config 2.0`
- **Livrable 4 :** résultats d'exécution de `poll_config 2.0`
- **Livrable 5 :** Fiche(s) rapport

1. Présentation

Le but de ce projet est de simuler un LAN en étoile dont la méthode d'accès au support est une libre interprétation de la méthode du polling.



Avec cette méthode, une station primaire (appelée aussi *maître*) invite séquentiellement des stations secondaires (appelées aussi *esclaves*) à émettre.

L'invitation à émettre n'est valable que durant un certain délai. A l'expiration de délai et **seulement à l'expiration de délai** la station primaire examine si elle a reçu une donnée provenant de la station secondaire invitée. Dans l'affirmative, elle diffuse cette donnée à toutes les autres stations, envoie un acquittement à la station émettrice et invite la station secondaire suivante. Sinon et sans attendre davantage, la station primaire invite à émettre, la station secondaire suivante.

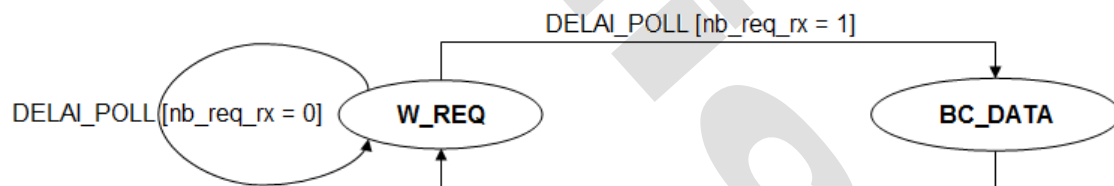
Si une station secondaire possède une donnée à émettre, elle l'émet vers la station primaire lorsqu'elle y a été invitée et attend l'acquittement de sa donnée. Sinon, elle ignore l'invitation.

Pour les besoins de la simulation, chaque station secondaire sera associée à un générateur de requêtes d'émission de données (*trafic* sur le schéma ci-dessus).

2. Description détaillée

Automate de la station primaire

Le comportement précis de la station primaire est décrit par l'automate ci-dessous. La station secondaire y est simplement appelée *station*. Chaque station possède un numéro i ($i \in [1 ; 5]$).



Evénements

- DELAJ_POLL : expiration du délai accordé à la station i pour émettre une donnée

Conditions

- $nb_req_rx = 0$: la station i invitée n'a aucune donnée à émettre
Action : afficher Prim ETAT_AUTOMATE Sti No_Data
- $nb_req_rx = 1$: la station i invitée a une donnée à émettre
Action : afficher Prim ETAT_AUTOMATE Sti Data_Rx

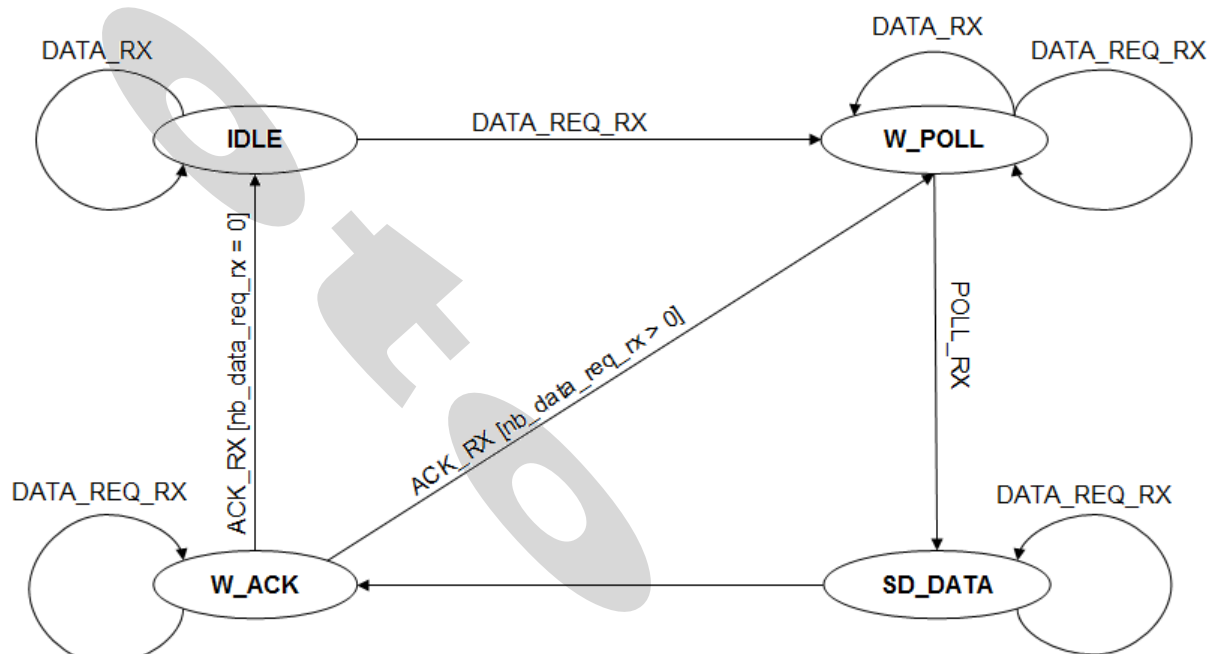
où i est le n° de la station secondaire invitée à émettre et ETAT_AUTOMATE le nom de l'état de l'automate dans lequel se trouve la station primaire

Etats

- W_REQ** : invitation de la station i à émettre et attente de sa réponse pendant DELAJ_POLL secondes.
Action en entrée : afficher Prim ETAT_AUTOMATE Sti POLL_DELAIs
- BC_DATA** : diffusion de la donnée de la station i à toutes les autres stations puis envoi d'un acquittement (ACK_TX) à la station i
Action en entrée : afficher Prim ETAT_AUTOMATE Sti

Automate de la station secondaire

Le comportement précis de la station secondaire est décrit par l'automate ci-dessous. La station secondaire y est simplement appelée *station*. Chaque station possède un numéro i ($i \in [1 ; 5]$).



Evénements

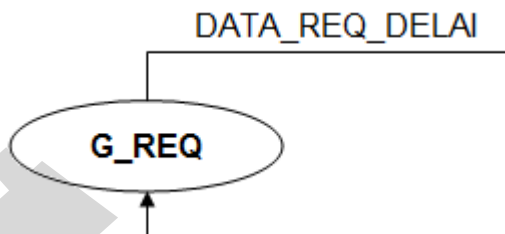
- **DATA_REQ_RX** : requête d'émission de donnée
Actions :
 1. incrémenter `nb_data_req_rx`
 2. afficher `TabSti ETAT_AUTOMATE Data_Req_Rx nb_data_req_rx`
où `Tab` représente $((3*i)+1)$ tabulations, i est le n° de la station secondaire et `nb_data_req_rx` la valeur courante de la variable `nb_data_req_rx`.
- **POLL_RX** : invitation à émettre une donnée.
NB: Cet événement n'est pris en compte que dans l'état **W_POLL**
Action : afficher `TabSti ETAT_AUTOMATE Poll_Rx`
- **ACK_RX** : acquittement de la dernière donnée émise
Actions :
 1. décrémenter `nb_data_req_rx`
 2. afficher `TabSti ETAT_AUTOMATE Ack_Rx nb_data_req_rx`Conditions :
 1. `nb_data_req_rx = 0` : la station n'a aucune requête d'émission de donnée en attente
 2. `nb_data_req_rx > 0` : la station a une ou plusieurs requêtes d'émission de donnée en attente
- **DATA_RX** : réception d'une donnée provenant d'une autre station
Action : afficher `TabSti ETAT_AUTOMATE Data_Rx`

Etats

- **IDLE** : la station n'a reçu aucune requête d'émission de donnée
Action lors de la 1^{ère} entrée ou en entrant depuis d'un autre état: afficher `TabSti ETAT_AUTOMATE`
- **W_POLL** : la station attend une invitation à émettre une donnée
Action en entrée depuis un autre état: afficher `TabSti ETAT_AUTOMATE Attente`
- **SD_DATA** : la station émet une donnée
Action en entrée depuis un autre état: afficher `TabSti ETAT_AUTOMATE`
- **W_ACK**: la station attend l'acquittement de la dernière donnée qu'elle a émise
Action en entrée depuis un autre état: afficher `TabSti ETAT_AUTOMATE`

Automate du générateur de requêtes d'émission de donnée

Le comportement précis du générateur est décrit par l'automate ci-dessous. La station secondaire y est simplement appelée *station*. Chaque station possède un numéro i ($i \in [1 ; 5]$).



Evénement

- **DATA_REQ_DELAI** : expiration du délai pour générer une requête d'émission de donnée vers sa station
Action : envoyer une requête d'émission de donnée vers sa station.

Etat

- **G_REQ** : attente de l'expiration du délai puis génération d'une requête d'émission de donnée.

3. Cahier des charges

3.1. Langage et bibliothèque de développement

Le projet sera développé en langage C. Les signaux seront gérés avec la bibliothèque POSIX. Vous devez utiliser les constantes, variables et fonction définies dans le fichier `polling.h` fourni en annexe 1.

3.2. Utilisation de la Forge

Pour Chaque groupe, un projet sera créé sur la Forge à :
https://projets3.ens.math-info.univ-paris5.fr/projects/soto_projets_l3_progunix_2017

3.3. Structure du simulateur

- L'automate d'une **station secondaire** sera implémenté dans un programme dont l'appel sera :
`secondaire numero_station pid_primaire`
où `numero_station` est le n° de la station secondaire et `pid_primaire` est le pid de la station primaire.
- L'automate de la **station primaire** sera implémenté dans un programme dont l'appel sera :
`primaire nb_polling delai_polling n pid_st1 ... pid_stn`
où `nb_polling` le nombre de cycle de polling des `n` stations secondaires, `delai_polling` est le délai pendant lequel la station primaire attend une réponse de la part de la station secondaire qu'elle a invitée à émettre, `n` ($n \in [1 ; 5]$) est le nombre de station secondaire et `pid_sti` le pid de la station secondaire n°i.
- L'automate du **générateur de requêtes d'émission de données** (DATA_REQ_RX) sera implémenté dans un programme dont l'appel sera :
`trafic i pid_Sti delai_min_requete delai_max_requete`
où `i` et `pid_Sti` sont respectivement le numéro et le pid de la station secondaire, `delai_min_requete` (> 0) et `delai_max_requete` ($\geq \text{delai_min_requete}$) sont respectivement le délai minimum et le délai maximum entre deux requêtes d'émission de données adressées à la station secondaire n°i pour laquelle `trafic` génère des requêtes d'émission de données. Ces deux délais sont les mêmes pour toutes les stations.
La graine de la séquence pseudo aléatoire des délais générés sera `i` pour la station secondaire n°i. Il y a un générateur pour chaque station secondaire.
- La **configuration et l'exécution du simulateur** seront implémentées dans un programme dont l'appel sera :
`./poll_config primaire secondaire trafic n nb_polling delai_polling delai_min_requete delai_max_requete [prefixe_fichier]`
où `n` ($n \in [1 ; 5]$) est le nombre de générateur de requêtes d'émission de donnée, de station secondaire à créer pour la simulation. Si le paramètre optionnel `prefixe_fichier` est présent, les résultats seront écrits dans un fichier nommé `prefixe_fichier_prim` pour la station primaire et `prefixe_fichier_sti` pour la station secondaire i. Sinon les résultats seront affichés à l'écran.

3.4. Affichages

TRES IMPORTANT:

Les affichages tels que définis dans les automates devront être respectés **au caractère et à la casse près, tabulations comprises**. Voir un exemple en annexe 2.

3.5. Développement

- **Automates**
Les automates de la station primaire, de la station secondaire et du générateur de trafic seront implémentés par une procédure nommée, respectivement, `primaire`, `secondaire` et `trafic`. Chacune de ces procédures utilisera la structure conditionnelle `switch` pour déterminer l'état courant de son automate.
- **Etats**
Les états seront décrits au moyen de `#define` (voir annexe 1).
Exemple :
`#define IDLE 1 // Aucune données à diffuser`

Les libellés des états seront implémentés au moyen de pointeurs de chaînes de caractères en respectant la convention de nommage : `string_nom-de-l'état` (voir annexe 1).

Exemple :

```
char *string_idle = "IDLE";
```

L'état courant d'un automate sera contenu dans la variable globale `int state` et son libellé dans la variable globale `char *string_state`.

- **Evènement**

Les **évènements** des différents automates seront implémentés au moyen des **signaux** et seront décrits au moyen de `#define` (voir annexe).

Exemple :

```
#define POLL_RX SIGUSR1 // Notification d'une invitation
                        // à émettre des données
```

Conventions

Selon qu'un même évènement est notifié ou généré, le `#define` correspondant sera suffixé, respectivement par `_RX` ou `_TX` (voir annexe) Cette convention ne s'applique pas aux évènements `DELAJ_POLL` et `DATA_REQ_DELAJ`.

Exemple :

```
#define POLL_RX      SIGUSR1 // Notification d'une invitation
                        // à émettre des données.

#define POLL_TX      SIGUSR1 // Génération d'une invitation
                        // à émettre des données
```


4. Travail demandé

4.1. Livrables

4.1.1. Le 04/12/2019 - Livrable 1 : Compréhension du problème

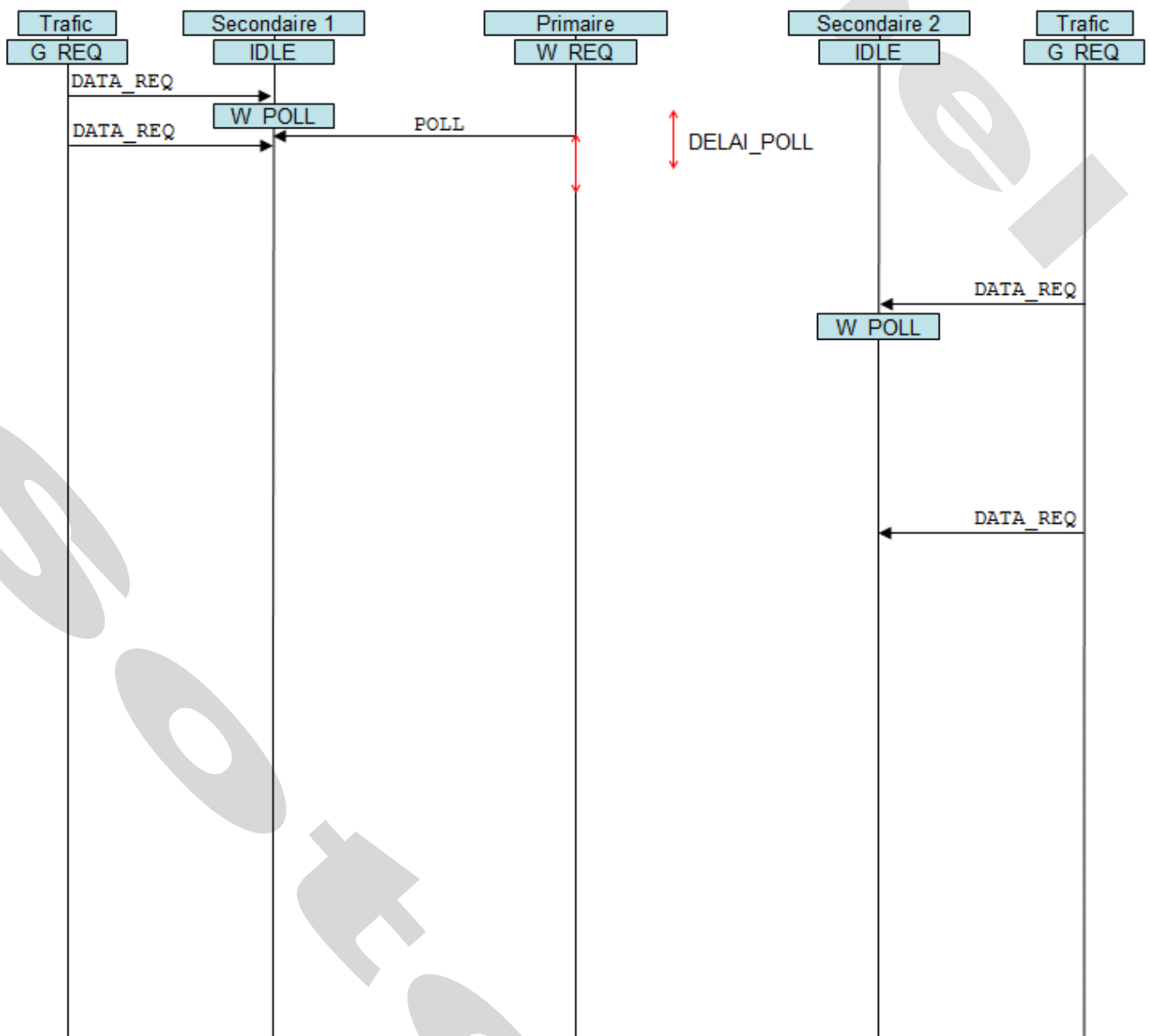
Complétez le diagramme de séquence des événements ci-dessous entre la station primaire et les stations secondaires 1 et 2 jusqu'à ce que les 2 stations secondaires se retrouvent dans l'état *Idle*.

Vous considérerez que le second DATA_REQ de la station secondaire 1 arrive alors que celle-ci est dans l'état *W_POLL* et que le second DATA_REQ de la station secondaire 2 arrive alors que celle-ci est dans l'état *SD_DATA*.

Une fois les deux stations dans l'état *Idle*, vous terminerez le diagramme lorsque la station primaire aura invité à émettre, à deux reprises, chacune des stations secondaires.

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Documents* de votre projet dans un fichier nommé `Liv1_PPUn` au format pdf où *n* est le n° de votre groupe de projet.



4.1.2. Le 12/12/2019- Livable 2 : code de `poll_config 1.0`

et de tous les programmes nécessaires à son exécution ainsi que de son `makefile`.

La partie simulation n'est pas implémentée dans ce livable. Il permet de créer tous les processus du simulateur avec une topologie en étoile selon ses paramètres d'appel et conformément au point 3.3 du cahier des charges.

Dans cette version, chaque processus affiche, pendant 60 secondes, qui il est (`poll_config`, `trafic`, `primaire`, `secondairei`), le nom et la valeur de ses paramètres d'exécution toujours conformément au cahier des charges.

Dépôt

Ce livable sera déposé sur la Forge sur l'onglet *Fichier* de votre projet dans un dossier compressé nommé `Liv2_PPUn` au format zip où `n` est le n° de votre groupe de projet.

4.1.3. Le 15/01/2020 - Livable 3 : code de `poll_config 2.0`

et de tous les programmes nécessaires à son exécution ainsi que de son `makefile`

Ce livable reprend le livable 2 et implémente les différents automates.

Dépôt

Ce livable sera déposé sur la Forge sur l'onglet *Fichiers* de votre projet dans un dossier compressé nommé `Liv3_PPUn` au format zip où `n` est le n° de votre groupe de projet.

4.1.4. Le 15/01/2020 - Livable 4 : résultats d'exécution de `poll_config 2.0`

Ce livable comprend les résultats d'exécutions suivants:

1. `poll_config primaire secondaire trafic 1 10 2 2 4 PPUn_1_10_2_2_4`
2. `poll_config primaire secondaire trafic 5 10 3 1 4 PPUn_5_10_3_1_5`

Dépôt

Ce livable sera déposé sur la Forge sur l'onglet *Fichiers* de votre projet dans un dossier compressé nommé `Liv4_PPUn` au format zip où `n` est le n° de votre groupe de projet.

4.1.5. 15/01/2020- Livable 5 : Fiche(s) rapport

Sur une ou plusieurs feuilles A4, vous écrirez tout ce que vous jugez nécessaire à l'évaluation de votre travail.

Dépôt

Ce livable sera déposé sur la Forge sur l'onglet *Documents* de votre projet dans un dossier compressé nommé `Liv5_PPUn` au format zip où `n` est le n° de votre groupe de projet.

5. Annexe 1

```
/*
 * polling.h
 *
 * Copyright 2016/09 Michel Soto
 *
 */
#ifndef EXIT_H

#define EXIT_H

#include <unistd.h>
#include <stdlib.h>

#endif

#define EVER (;;)
#define MAX_ST 5 // Nombre maximal de station

#define POLL_RX SIGUSR1 // Arrivée d'une invitation à émettre des données
#define POLL_TX SIGUSR1 // Envoi d'une invitation à émettre des données
#define DATA_RX SIGUSR2 // Réception de données
#define DATA_TX SIGUSR2 // Emission de données
#define ACK_RX SIGPIPE // Réception acquittement de données émises
#define ACK_TX SIGPIPE // Envoi acquittement de données émises
#define DATA_REQ_RX SIGALRM // Arrivée d'une requête d'émission de données
#define DATA_REQ_TX SIGALRM // Envoi d'une requête d'émission de données

// Valeur numériques associées aux états de automates
#define IDLE 1 // Aucune données à diffuser
#define W_REQ 2 // WAIT_REQUEST: Attente d'une demande d'émission
// de données de la part de la station i
#define W_POLL 3 // WAIT_POLL: Attente d'une invitation à émettre
// des données de la part de la station primaire
#define W_ACK 4 // WAIT_ACK: Attente d'un acquittement de données
// de la part de la station primaire
#define BC_DATA 5 // BROADCASTING_DATA: Diffusion de données
#define SD_DATA 6 // SENDING_DATA: Envoi d'une donnée
#define G_REQ 7 // GENERATING_REQUEST: Génération d'un requête
// d'envoi de donnée

// Chaînes de caractères associées aux états de automates
char *string_idle = "IDLE";
char *string_g_req = "G_REQ";
char *string_w_req = "W_REQ";
char *string_w_ack = "W_ACK";
char *string_w_poll = "W_POLL";
char *string_bc_data = "BC_DATA";
char *string_sd_data = "SD_DATA";

// -----
void erreur (char *message) {
// -----
    perror(message);
    exit (EXIT_FAILURE);
} // erreur
```

6. Annexe 2

```
St1 IDLE
Prim W_REQ St1 2s
St2 IDLE
St2 IDLE Data_Req_Rx 1
St2 W_POLL Attente
St1 IDLE Data_Req_Rx 1
St1 W_POLL Attente
Prim W_REQ St1 No_Data
Prim W_REQ St2 2s
St2 W_POLL Poll_Rx
St2 SD_DATA
St2 W_ACK
Prim W_REQ St2 Data_Rx
Prim BC_DATA St2
Prim W_REQ St1 2s
St2 W_ACK Ack_Rx 0
St2 IDLE
St1 W_POLL Data_Rx
St1 W_POLL Poll_Rx
St1 SD_DATA
St1 W_ACK
St1 W_ACK Data_Req_Rx 2
St2 IDLE Data_Req_Rx 1
St2 W_POLL Attente
Prim W_REQ St1 Data_Rx
Prim BC_DATA St1
Prim W_REQ St2 2s
St2 W_POLL Data_Rx
St2 W_POLL Poll_Rx
St1 W_ACK Ack_Rx 1
St1 W_POLL Attente
St2 SD_DATA
St2 W_ACK
St2 W_ACK Data_Req_Rx 2
Prim W_REQ St2 Data_Rx
Prim BC_DATA St2
```