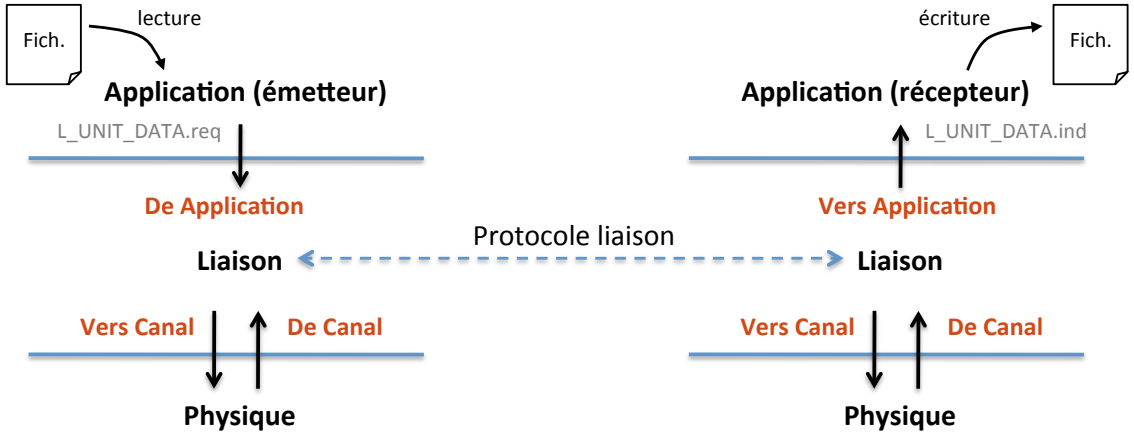
Projet réseaux : développement et test d'une couche liaison de données point-à-point

1 Présentation générale du projet

L'objectif de ce projet est de développer et tester des protocoles de communication point-à-point mis en œuvre au sein d'une couche liaison de données (le langage C sera utilisé). Dans la suite du projet, nous supposons un modèle théorique en 3 couches (identique à celui vu en TD) : physique, liaison de données et application.

- La **couche physique** a déjà été implantée et vous est fournie sous la forme d'un module. Elle simule le comportement d'une couche physique. Elle est détaillée dans la section 4.
- La **couche liaison de données** constitue le cœur du projet. Vous devrez développer dans cette couche plusieurs protocoles de liaison point-à-point (unidirectionnels) supportant différents types de services détaillés dans la section 3.
- La **couche application** intègre un protocole de transfert de fichiers entre un émetteur et un récepteur. Cette couche a également été développée, elle vous est fournie sous la forme d'un module détaillé dans la section 2.

La figure ci-dessous illustre l'articulation de ces trois couches (ici l'application utilise le service liaison `L_UNIT_DATA` en mode non confirmé).



2 Couche application

Un protocole de transfert de fichiers entre un émetteur et un récepteur est implanté dans la couche application. L'émetteur lit un fichier par blocs de données, chacun d'eux est remis à la couche liaison pour transmission. Le récepteur récupère les blocs de données de sa couche liaison et les écrit dans le fichier de destination. Il existe deux versions de la couche application :

- `couche_appli_non_connectee.c` dans laquelle l'application s'appuie sur les services de transmission de la couche liaison en mode non connecté.
- `couche_appli_connectee.c` dans laquelle l'application s'appuie sur les services de transmission de la couche liaison en mode connecté.

L'interface de programmation entre la couche liaison de données et la couche application est spécifiée dans le fichier `application.h` et contient deux fonctions : `de_application` pour récupérer des données ou indications issues de la couche application et `vers_application` pour les lui remettre.

```
/*************************************
 * Interface avec la couche application *
 *************************************/

/*
 * Lecture de données émanant de la couche application. Ces données
 * doivent être traitées selon un certain service exigé par l'application
 * via l'argument service_liaison.
 * Paramètres (en sortie):
 *   - service_liaison : service demandé par l'application
 *                       (cf. primitives de services_liaison.h)
 *   - donnees : message devant être envoyé (issu de la couche application)
 *   - taille_msg : nombre d'octets de données à émettre (0 si aucune donnée)
 */
void de_application(int *service_liaison, char *donnees, int *taille_msg);

/*
 * Remise de données à la couche application.
 * Paramètres (en entrée):
 *   - service_liaison : service fourni par la couche liaison
 *                       (cf. primitives de services_liaison.h)
 *   - donnees : données reçues par la liaison, remontées à l'application
 *   - taille_msg : taille des données reçues
 * Renvoie :
 *   - En mode non connecté, avec sercice_liaison = L_UNIT_DATA_ind
 *      -> 1 si l'appli réceptrice souhaite terminer le service de liaison
 *      -> 0 sinon
 *   - En mode connecté, avec service_liaison = L_CONNECT_ind
 *      -> La réponse à la demande de connexion
 *         (L_CONNECT_resp_ACCEPT ou L_CONNECT_resp_REFUSE)
 */
int vers_application(int service_liaison, char *donnees, int taille_msg);
```

3 Spécification de la couche liaison de données

3.1 Services offerts à la couche application, et protocoles sous-jacents

Service liaison non connecté, non fiable

Le service est rendu en mode non connecté (phase unique de transfert de données). Aucune garantie n'est apportée sur la délivrance des données (des erreurs ou pertes peuvent se produire).

▶ **Protocole liaison 1** *Mode non connecté, sans contrôle de flux, ni contrôle et reprise sur erreurs.*

▶ **Protocole liaison 2** *Mode non connecté avec uniquement contrôle de flux de type “Stop and Wait” ; pas de contrôle et reprise sur erreurs.*

Service liaison non connecté, fiable

Le service est rendu en mode non connecté (phase unique de transfert de données). Cependant, des garanties de fiabilité sont apportées : le service gère les erreurs et les pertes (à noter que cela est plutôt un scénario pédagogique, dans la grande majorité des cas le mode non connecté est non fiable).

▶ **Protocole liaison 3** *Mode non connecté avec contrôle de flux et reprise sur erreurs de type "Stop and Wait PAR". Le code détecteur d'erreurs utilisera une somme de contrôle calculée sur tous les octets de données de la trame en appliquant l'opérateur "ou exclusif" (XOR).*

Service liaison connecté, fiable

Le service est rendu en mode connecté (trois phases successives : établissement de la connexion, transfert de données, terminaison de la connexion). La communication est fiable : les erreurs et pertes sont gérées.

▶ **Protocole liaison 4** *Mode connecté appliqué au protocole liaison 3 (contrôle de flux et reprise sur erreurs de type "Stop and Wait PAR"). On veillera en particulier à la fiabilité des échanges liés à l'établissement et à la libération de la connexion.*

▶ **Protocole liaison 5** *Mode connecté avec fenêtre d'anticipation et utilisant un mécanisme de reprise sur erreurs de type "Go Back N" (la taille de la fenêtre d'émission sera de 7).*

3.2 Primitives de services (`services_liaison.h`)

Sont listées ci-dessous les différentes primitives de services de la couche liaison de données (la terminologie utilisée est issue du modèle de référence OSI).

```
/************************************************
 * Primitives de service pour le mode non connecté *
 ************************************************/

/* transfert d'une unité de données */
#define L_UNIT_DATA_req  1
/* notification de la réception d'une unité de données */
#define L_UNIT_DATA_ind  2

/********************************************
 * Primitives de service pour le mode connecté *
 ********************************************/

/* demande d'établissement de connexion */
#define L_CONNECT_req           10
/* notification d'une demande d'établissement de connexion */
#define L_CONNECT_ind           11
/* réponse à la demande de connexion : acceptation */
#define L_CONNECT_resp_ACCEPT   12
/* réponse à la demande de connexion : refus */
#define L_CONNECT_resp_REFUSE   13
/* notifier la réponse à la demande de connexion : acceptation */
#define L_CONNECT_conf_ACCEPT   14
/* notifier la réponse à la demande de connexion : refus */
#define L_CONNECT_conf_REFUSE   15
/* transfert d'une unité de données au sein d'une connexion */
#define L_DATA_req              16
/* notif. de la réception d'une unité de données au sein d'une connex. */
#define L_DATA_ind              17
/* fermeture de connexion */
#define L_DISCONNECT_req        18
/* notification de fermeture de connexion */
#define L_DISCONNECT_ind        19
```

3.3 Unités de données des protocoles liaison (`couche_liaison.h`)

Les unités de données des protocoles liaison (appelées *trame*) sont spécifiées dans cette section. Pour simplifier, le même format de trame sera utilisé pour tous les protocoles décrits ci-dessus. Certains protocoles n'utiliseront donc pas tous les champs de la structure de la trame.

```
/************************
 * Structure d'une trame *
 ************************/
typedef struct trame_s {
  uint8_t type;       /* type de trame, cf. ci-dessous */
  uint8_t num_seq;    /* numéro de séquence */
  uint8_t lg_info;    /* longueur du champ info */
  char info[MTU];     /* données utiles de la trame */
  char fcs;           /* somme de contrôle */
} trame_t;

/******************
 * Types de trame *
 ******************/
#define CON_REQ       0  /* demande d'établissement de connexion */
#define CON_ACCEPT    1  /* acceptation de connexion */
#define CON_REFUSE    2  /* refus d'établissement de connexion */
#define CON_CLOSE     3  /* notification de déconnexion */
#define CON_CLOSE_ACK 4  /* accusé de réception de la déconnexion */
#define DATA          5  /* données de l'application */
#define ACK           6  /* accusé de réception des données */
#define OTHER         7  /* extensions */
```

Note : on pensera également à ajouter dans le module `couche_liaison.h` des fonctions utilitaires qui pourront être exploitées par certains protocoles liaison (`generer_controle`, etc.).

4 Utilisation des services de la couche physique (`services_physique.h`)

Vous disposez d'une bibliothèque (`services_physique.h`) qui simule le comportement d'une couche physique. Pour utiliser cette bibliothèque, vous devez commencer par appeler la fonction d'initialisation `init_physique()`. Puis, vous utilisez les primitives `vers_` et `de_canal` pour envoyer et recevoir des trames. Enfin, vous disposez de fonctions utilitaires pour la gestion de minuteurs (*timers*) et l'attente d'un événement (*timeout* ou trame arrivée).

Initialisation couche physique

```
/*********************************************************
 * Initialisation de le couche physique. Le paramètre rôle *
 * doit être égal à 1 pour la réception, 0 pour l'émission. *
 *********************************************************/
void init_physique(int role);
```

Primitives de service pour émission et réception sur le canal

```
/*********************************************************
 * Remet une trame à la couche physique pour émission     *
 * sur le support de communication.                       *
 *********************************************************/
void vers_canal(trame_t *trame);
```

```
/*********************************************************
 * Prélève une trame de la couche physique (N.B. : fonction  *
 * bloquante tant qu'une trame n'est pas reçue).             *
 *********************************************************/
void de_canal(trame_t *trame);
```

Fonctions utilitaires pour la gestion des temporisateurs (*timers*)

```
/*************************************************************
 * Démarre le timer numéro n (0 < n < 100) qui s'arrête après  *
 * ms millisecondes (ms doit être un multiple de 100)          *
 *************************************************************/
void depart_temporisateur(int n, int ms);

/*****************************************
 * Arrête le timer numéro n (0 < n < 100)  *
 *****************************************/
void arreter_temporisateur(int n);

/*********************************************************
 * Test si le timer numéro n (0 < n < 100) est en marche  *
 * Renvoie :  1 si le timer numéro n est en route         *
 *            0 sinon                                     *
 *********************************************************/
int test_temporisateur(int n);

/*********************************************************************
 * Fonction qui attend un évènement (trame reçue ou timeout).        *
 * (N.B. : fonction bloquante)                                       *
 * Renvoie :  0 si une trame reçue est disponible,                   *
 *            un numéro de timer [1-100] si un timeout a été généré  *
 *********************************************************************/
int attendre();
```

5 Travail à réaliser

Génération des exécutables `emetteur` et `recepteur`

Le travail demandé consiste à implanter les protocoles (1 à 5) de la couche liaison de données énoncés dans la section 3.1. Pour chaque cas, vous développerez un programme principal pour l'émetteur et un programme principal pour le récepteur, qui devront respecter la convention de nommage suivante (exemple avec le protocole liaison 1) :

- `proto_liaison_v1_emetteur.c` : *main()* de l'émetteur pour le protocole liaison 1.
- `proto_liaison_v1_recepteur.c` : *main()* du récepteur pour le protocole liaison 1.

Nous vous fournissons un *Makefile* pour effectuer la compilation et l'édition de liens. Exemple d'un cycle de travail classique :

1. `make clean` : suppression des exécutables et fichiers objets.
2. `make sl1` pour générer les exécutables du protocole liaison 1 dans dossier `bin/`
3. (correction du code source si erreurs éventuelles jusqu'au succès du *make*)
4. Edition des paramètres de configuration dans `config.txt` (cf. ci-dessous).

5. Test : `bin/recepteur` pour exécuter le processus de réception et `bin/emetteur` pour exécuter le processus d'émission (dans un terminal différent !).
6. Comparez les fichiers émis et reçus...

Fichier de configuration `config.txt`

Le fichier `config.txt` contient tous les paramètres de configuration de l'application. **Il est important de faire varier ces paramètres afin de tester vos protocoles**.

Paramètres qui configurent le nom des fichiers manipulés par la couche application :
- `FICHIER_IN` indique le fichier lu par l'émetteur.
- `FICHIER_OUT` indique le nom du fichier dans lequel seront écrites les données reçues.

Paramètres qui contrôlent le comportement de la couche physique :
- `PROBA_PERTE_E` et `PROBA_PERTE_R` pour indiquer le taux de pertes respectivement côté émetteur et récepteur. Cette valeur est comprise entre 0 et 1 (notez qu'une probabilité au dessus de 0.5 est déconseillée).
- `PROBA_ERREUR_E` et `PROBA_ERREUR_R` pour indiquer le taux d'erreurs respectivement côté émetteur et récepteur. Cette valeur est comprise entre 0 et 1 (notez qu'une probabilité au dessus de 0.5 est déconseillée).
- `PERTE_CON_REQ` et `PERTE_CON_ACCEPT` pour forcer une perte respectivement de la demande de connexion et de l'acceptation d'une demande de connexion. Cette valeur vaut 0 (pas de perte forcée) ou 1 (perte).
- `PERTE_CON_CLOSE` et `PERTE_CON_CLOSE_ACK` pour forcer une perte respectivement de la déconnexion et de l'accusé de réception de cette déconnexion. Cette valeur vaut 0 (pas de perte forcée) ou 1 (perte).

6 Evaluation

Le projet est à réaliser individuellement. Vous disposez de 5 séances de TP, la validation se fera lors de la 6ème séance.

Lors de la validation, vous ferez une démonstration à l'enseignant du transfert de fichier via les différents protocoles de liaison que vous avez implantés. Ce même jour vous devrez également remettre à votre enseignant de TP votre code source commenté (archive des fichiers au format .zip ou .tar.gz par courrier électronique ou dépôt sur moodle).