```
// liaison.cc
// version modifiee (Avec Trame_NUM [V5])

#include "copyright.h"
#include "liaison.h"
#include "system.h"
#include <unistd.h>

#include <strings.h>

#define LARGEUR_FENETRE_EMISSION  5
#define LARGEUR_FENETRE_RECEPTION 2

// Les 3 fonctions suivantes servent juste à rediriger l'exécution des
// Threads correspondants vers les bonnes fonctions C++...

static void
ProtocoleReception_Root (int arg)
{
  ((CoucheLiaison *) arg)->ProtocoleReception ();
}

static void
ProtocoleEmission_Root (int arg)
{
  ((CoucheLiaison *) arg)->ProtocoleEmission ();
}

static void
DemonReception_Root (int arg)
{
  ((CoucheLiaison *) arg)->DemonReception ();
}


CoucheLiaison::CoucheLiaison (NetworkAddress addr, double reliability)
{
  Thread *tr, *ts, *tf;

  couchePhysique = new CouchePhysique (addr, reliability);

  tamponReception = new Tampon (LARGEUR_FENETRE_RECEPTION);
  tamponEmission = new Tampon (LARGEUR_FENETRE_EMISSION);
  tamponDemon = new SynchList ();

  temporisateur = new Temporisateur;

  tr = new Thread ("protocole de reception");
  tr->Fork (ProtocoleReception_Root, (int) this);

  ts = new Thread ("protocole d'émission");
  ts->Fork (ProtocoleEmission_Root, (int) this);

  tf = new Thread ("demon de réception");
  tf->Fork (DemonReception_Root, (int) this);
}

CoucheLiaison::~CoucheLiaison ()
{
  delete couchePhysique;
  delete tamponReception;
  delete tamponEmission;
  delete tamponDemon;
  delete temporisateur;
}

void
CoucheLiaison::EnvoyerPaquet (char *data, unsigned len)
{
```



```
  Trame *trame = new Trame (Trame_INFO, data, len);

  tamponEmission->Ajouter (trame);
  evtEmission.Generer (new Evenement (Event_NEW_FRAME));
}

void
CoucheLiaison::RecevoirPaquet (char *data, unsigned *len)
{
  Trame *trame;

  trame = tamponReception->Retirer ();

  bcopy (trame->data, data, trame->header.len);
  *len = trame->header.len;

  delete trame;

  evtReception.Generer (new Evenement (Event_FRAME_USED)); // [V5]
}

void
CoucheLiaison::Emettre (unsigned trame_courante)
{
  Trame *trame;

  // On récupère la trame du tampon d'émission
  trame = tamponEmission->TrameNumero (trame_courante);

  trame->header.NS = trame_courante;
  trame->header.DebugInfo = trame_courante; // inutile au protocole

  // On l'envoie sur le réseau
  DEBUG ('l', "\t [E] Emission de la trame %d\n", trame_courante);
  couchePhysique->EnvoyerTrame (trame);

  temporisateur->Armer (2 * TEMPO_ACQUITTEMENT, Expiration_ACK,
                        trame_courante, &evtEmission);
}

void
CoucheLiaison::ProtocoleEmission ()
{

  Evenement *evt;
  unsigned ack_attendu = 0;  // numéros de séquence à usage local
  unsigned trame_courante = 0;
  unsigned trame_limite = 1; // [V5] - on suppose qu'il y a au moins une place d
e libre au demarrage
  unsigned trame_limite_tampon_emission = 0; // [V5]

  for (;;)
    {

      evt = evtEmission.Recuperer ();   // On attend le prochain
                                        // événement
      switch (evt->type)
        {

        case Event_NEW_FRAME:
          {
            // La couche supérieure a réussi a déposer une trame
            // supplémentaire dans le tampon de
            // LARGEUR_FENETRE_EMISSION elt

            // [V5]
            if (trame_courante==0) // on amorce les envois
              {
                Emettre(0);
```

```
                trame_courante++;
              }
            trame_limite_tampon_emission++;

            break;
          }

        case Event_NUM_RECEIVED: // [V5]
          {
            DEBUG ('l', "\t [E] Trame NUM reçue \n");

            Trame *trame_num = (Trame *) tamponDemon->Remove ();
            trame_limite = trame_num->header.NR + trame_num->header.libre; // mi
se a jour de trame_limite
            // emission en rafale
            for (;(trame_courante<=trame_limite) && (trame_courante<trame_limite
_tampon_emission); trame_courante++)
              Emettre(trame_courante);
            break;
          }

        case Event_ACK_RECEIVED:
          {

            Trame *trame_ack = (Trame *) tamponDemon->Remove ();

            if (trame_ack->header.NR == ack_attendu)
              {

                // On reçoit un acquittement
                DEBUG ('l', "\t [E] Acquittement reçu pour la trame %d\n",
                       ack_attendu);
                DEBUG ('l', "\t [E] Trame limite = %d\n", trame_limite); // [V5]

                ack_attendu++;

                // On peut retirer la trame émise du tampon et la
                // détruire...

                delete tamponEmission->Retirer ();
              }

            else
              {
                // C'est un acquittement reçu "hors-séquence"
                DEBUG ('l', "\t [E] Acquittement %d ignoré (attendu %d) \n",
                       trame_ack->header.NR, ack_attendu);
              }
            delete trame_ack;
            break;
          }

        case Event_TIMER_EXPIRATION:
          {
            // expiration d'un temporisateur

            if (evt->num >= ack_attendu)
              {

                DEBUG ('l',
                       "\t [E] Expiration du temporisateur pour la trame %d\n",
                       evt->num);

                // Il faut ré-émettre perdue...
                Emettre (evt->num);
              }

            break;
          }
```



```
        default:
          {
            fprintf (stderr, "Oups: événement inattendu en émission !\n");
            interrupt->Halt ();
          }
        }                           // switch

      delete evt;
    }
}

void
CoucheLiaison::ProtocoleReception ()
{
  Evenement *evt;
  unsigned numero_attendu = 0;
  unsigned numero_recu = 0; // [V5]

  for (;;)
    {

      // On attend le prochain événement
      evt = evtReception.Recuperer ();

      switch (evt->type)
        {

        case Event_NEW_FRAME:
          {

            Trame *trame_ack = new Trame (Trame_ACK);
            Trame *trame_recue = (Trame *) tamponDemon->Remove ();

            trame_ack->header.NR = trame_recue->header.NS;
            trame_ack->header.DebugInfo = trame_recue->header.DebugInfo;

            numero_recu=trame_ack->header.NR; // [V5]

            if (trame_recue->header.NS == numero_attendu)
              {
                if(tamponReception->TenterAjout (trame_recue) == -1)
                  {
                    // Oups! Le tampon est plein...
                    DEBUG ('l', "\t [R] Trame %d détruite (tampon saturé)\n",
                           trame_recue->header.NS);
                    delete trame_recue;

                    break; // pour ne pas envoyer d'ack
                  }
                numero_attendu++;
              }
            else
              {
                DEBUG ('l', "\t [R] Trame %d ignorée (attendue %d)\n",
                       trame_recue->header.NS, numero_attendu);
                delete trame_recue;
              }

            DEBUG ('l', "\t [R] Emission d'un ack %d (trame %d)\n",
                   trame_ack->header.NR, trame_ack->header.DebugInfo);
            couchePhysique->EnvoyerTrame (trame_ack);
            break;
          }

        case Event_FRAME_USED: // [V5]
          {
            DEBUG ('l', "\t [R] Une trame a ete consommee: emission d'une trame NUM \n");
            Trame *trame_num = new Trame (Trame_NUM);
```

```
            trame_num->header.NR=numero_recu;
            trame_num->header.libre=LARGEUR_FENETRE_RECEPTION - tamponReception-
>NbTrames(); // [V5] : calcul du nombre de places libres dans le tampon du recep
teur
            couchePhysique->EnvoyerTrame (trame_num);
            break;
          }

        default:
          {
            fprintf (stderr, "Oups: événement inattendu en réception !\n");
            interrupt->Halt ();
          }

        }

      delete evt;
    }
}

void
CoucheLiaison::DemonReception ()
{

  for (;;)
    {

      Trame *trame = couchePhysique->RecevoirTrame ();

      tamponDemon->Append (trame);

      switch (trame->header.type)
        {
        case Trame_INFO:
          {

            DEBUG ('l', "\t [D] Réception de la trame %d \n",
                   trame->header.DebugInfo);

            evtReception.
              Generer (new Evenement (Event_NEW_FRAME));

            break;
          }
        case Trame_ACK:
          {
            DEBUG ('l', "\t [D] Réception de l'ack %d (trame %d) \n",
                   trame->header.NR, trame->header.DebugInfo);

            // On signale l'événement au démon de réception
            evtEmission.
              Generer (new Evenement (Event_ACK_RECEIVED));
            break;
          }

        case Trame_REJECT:
          {
            ASSERT (FALSE);
            break;
          }

        case Trame_NUM: // [V5]
          {
            DEBUG ('l', "\t [D] Récepion d'une trame NUM\n");
            // On signale l'évènement au démon d'émission (protocole)
            evtEmission.Generer (new Evenement (Event_NUM_RECEIVED));
            break;
          }
```



```
        default:
          {
            fprintf (stderr, "Oups: trame de type inconnu reçue !\n");
            interrupt->Halt ();
          }
        }                           // switch

    }                               // for
}
```

```
// liaison.cc
// version modifiee (Avec Trame_NUM [V5] et TIMER_NUM [V6])

#include "copyright.h"
#include "liaison.h"
#include "system.h"
#include <unistd.h>

#include <strings.h>

#define LARGEUR_FENETRE_EMISSION  5
#define LARGEUR_FENETRE_RECEPTION 2

// Les 3 fonctions suivantes servent juste à rediriger l'exécution des
// Threads correspondants vers les bonnes fonctions C++...

static void
ProtocoleReception_Root (int arg)
{
  ((CoucheLiaison *) arg)->ProtocoleReception ();
}

static void
ProtocoleEmission_Root (int arg)
{
  ((CoucheLiaison *) arg)->ProtocoleEmission ();
}

static void
DemonReception_Root (int arg)
{
  ((CoucheLiaison *) arg)->DemonReception ();
}


CoucheLiaison::CoucheLiaison (NetworkAddress addr, double reliability)
{
  Thread *tr, *ts, *tf;

  couchePhysique = new CouchePhysique (addr, reliability);

  tamponReception = new Tampon (LARGEUR_FENETRE_RECEPTION);
  tamponEmission = new Tampon (LARGEUR_FENETRE_EMISSION);
  tamponDemon = new SynchList ();

  temporisateur = new Temporisateur;
  temporisateur_NUM = new Temporisateur;

  tr = new Thread ("protocole de reception");
  tr->Fork (ProtocoleReception_Root, (int) this);

  ts = new Thread ("protocole d'émission");
  ts->Fork (ProtocoleEmission_Root, (int) this);

  tf = new Thread ("demon de réception");
  tf->Fork (DemonReception_Root, (int) this);
}

CoucheLiaison::~CoucheLiaison ()
{
  delete couchePhysique;
  delete tamponReception;
  delete tamponEmission;
  delete tamponDemon;
  delete temporisateur;
}

void
CoucheLiaison::EnvoyerPaquet (char *data, unsigned len)
```



```
{
  Trame *trame = new Trame (Trame_INFO, data, len);

  tamponEmission->Ajouter (trame);
  evtEmission.Generer (new Evenement (Event_NEW_FRAME));
}

void
CoucheLiaison::RecevoirPaquet (char *data, unsigned *len)
{
  Trame *trame;

  trame = tamponReception->Retirer ();

  bcopy (trame->data, data, trame->header.len);
  *len = trame->header.len;

  delete trame;

  evtReception.Generer (new Evenement (Event_FRAME_USED)); // [V5]
}

void
CoucheLiaison::Emettre (unsigned trame_courante)
{
  Trame *trame;

  // On récupère la trame du tampon d'émission
  trame = tamponEmission->TrameNumero (trame_courante);

  trame->header.NS = trame_courante;
  trame->header.DebugInfo = trame_courante; // inutile au protocole

  // On l'envoie sur le réseau
  DEBUG ('l', "\t [E] Emission de la trame %d\n", trame_courante);
  couchePhysique->EnvoyerTrame (trame);

  temporisateur->Armer (2 * TEMPO_ACQUITTEMENT, Expiration_ACK,
                        trame_courante, &evtEmission);
}

void
CoucheLiaison::ProtocoleEmission ()
{

  Evenement *evt;
  unsigned ack_attendu = 0;  // numéros de séquence à usage local
  unsigned trame_courante = 0;
  unsigned trame_limite = 1; // [V5] - on suppose qu'il y a au moins une place d
e libre au demarrage
  unsigned trame_limite_tampon_emission = 0; // [V5]

  for (;;)
    {

      evt = evtEmission.Recuperer ();   // On attend le prochain
                                        // événement
      switch (evt->type)
        {

        case Event_NEW_FRAME:
          {
            // La couche supérieure a réussi a déposer une trame
            // supplémentaire dans le tampon de
            // LARGEUR_FENETRE_EMISSION elt

            // [V5]
            if (trame_courante==0) // on amorce les envois
              {
```

```
                Emettre(0);
                trame_courante++;
              }
            trame_limite_tampon_emission++;

            break;
          }

        case Event_NUM_RECEIVED: // [V5]
          {
            DEBUG ('l', "\t [E] Trame NUM reçue \n");

            Trame *trame_num = (Trame *) tamponDemon->Remove ();
            trame_limite = trame_num->header.NR + trame_num->header.libre; // mi
se a jour de trame_limite
            // emission en rafale
            for (;(trame_courante<=trame_limite) && (trame_courante<trame_limite
_tampon_emission); trame_courante++)
              Emettre(trame_courante);
            break;
          }

        case Event_ACK_RECEIVED:
          {

            Trame *trame_ack = (Trame *) tamponDemon->Remove ();

            if (trame_ack->header.NR == ack_attendu)
              {

                // On reçoit un acquittement
                DEBUG ('l', "\t [E] Acquittement reçu pour la trame %d\n",
                       ack_attendu);
                DEBUG ('l', "\t [E] Trame limite = %d\n", trame_limite); // [V5]

                ack_attendu++;

                // On peut retirer la trame émise du tampon et la
                // détruire...

                delete tamponEmission->Retirer ();
              }

            else
              {
                // C'est un acquittement reçu "hors-séquence"
                DEBUG ('l', "\t [E] Acquittement %d ignoré (attendu %d) \n",
                       trame_ack->header.NR, ack_attendu);
              }
            delete trame_ack;
            break;
          }

        case Event_TIMER_EXPIRATION:
          {
            // expiration d'un temporisateur

            if (evt->num >= ack_attendu)
              {

                DEBUG ('l',
                       "\t [E] Expiration du temporisateur pour la trame %d\n",
                       evt->num);

                // Il faut ré-émettre la trame perdue...
                Emettre (evt->num);
              }

            break;
```



```
          }

        default:
          {
            fprintf (stderr, "Oups: événement inattendu en émission !\n");
            interrupt->Halt ();
          }
        }                          // switch

      delete evt;
    }
}

void
CoucheLiaison::ProtocoleReception ()
{
  Evenement *evt;
  unsigned numero_attendu = 0;
  unsigned numero_recu = 0; // [V5]

  for (;;)
    {

      // On attend le prochain événement
      evt = evtReception.Recuperer ();

      switch (evt->type)
        {

        case Event_NEW_FRAME:
          {

            Trame *trame_ack = new Trame (Trame_ACK);
            Trame *trame_recue = (Trame *) tamponDemon->Remove ();

            trame_ack->header.NR = trame_recue->header.NS;
            trame_ack->header.DebugInfo = trame_recue->header.DebugInfo;

            numero_recu=trame_ack->header.NR; // [V5]

            if (trame_recue->header.NS == numero_attendu)
              {
                if(tamponReception->TenterAjout (trame_recue) == -1)
                  {
                    // Oups! Le tampon est plein...
                    DEBUG ('l', "\t [R] Trame %d détruite (tampon saturé)\n",
                           trame_recue->header.NS);
                    delete trame_recue;

                    break; // pour ne pas envoyer d'ack
                  }
                numero_attendu++;
              }
            else
              {
                DEBUG ('l', "\t [R] Trame %d ignorée (attendue %d)\n",
                       trame_recue->header.NS, numero_attendu);
                delete trame_recue;
              }

            DEBUG ('l', "\t [R] Emission d'un ack %d (trame %d)\n",
                   trame_ack->header.NR, trame_ack->header.DebugInfo);
            couchePhysique->EnvoyerTrame (trame_ack);
            break;
          }

        case Event_FRAME_USED: // [V5]
          {
            DEBUG ('l', "\t [R] Une trame a ete consommee: emission d'une trame NUM \n");
```

```
            Trame *trame_num = new Trame (Trame_NUM);
            trame_num->header.NR=numero_recu;
            trame_num->header.libre=LARGEUR_FENETRE_RECEPTION - tamponReception-
>NbTrames();
            // [V5] : calcul du nombre de places libres dans le tampon du recept
eur

            couchePhysique->EnvoyerTrame (trame_num);
            temporisateur_NUM->Armer (2 * TEMPO_NUM, Expiration_NUM,
                                      0, &evtReception); // [V6] : on arme un te
mporisateur
            break;
          }

        case Event_TIMER_NUM_EXPIRATION: // [V6]
          {
            // expiration du temporisateur pour trame NUM
            if (LARGEUR_FENETRE_RECEPTION - tamponReception->NbTrames() > 0) {
              DEBUG ('l', "\t [R] Expiration du temporisateur NUM: ré–émission de la trame NUM \n");
              Trame *trame_num = new Trame (Trame_NUM);
              trame_num->header.NR=numero_recu;
              trame_num->header.libre=LARGEUR_FENETRE_RECEPTION - tamponReceptio
n->NbTrames();
              // On met à jour les places libres
              couchePhysique->EnvoyerTrame (trame_num);
              temporisateur->Armer (2 * TEMPO_NUM, Expiration_NUM,
                                    0, &evtReception); // [V6] : on arme un nouv
eau temporisateur
            }

            break;
          }

        default:
          {
            fprintf (stderr, "Oups: événement inattendu en réception !\n");
            interrupt->Halt ();
          }

        }

      delete evt;
    }
}

void
CoucheLiaison::DemonReception ()
{

  for (;;)
    {

      Trame *trame = couchePhysique->RecevoirTrame ();

      tamponDemon->Append (trame);

      switch (trame->header.type)
        {
        case Trame_INFO:
          {

            DEBUG ('l', "\t [D] Réception de la trame %d \n",
                   trame->header.DebugInfo);

            evtReception.
              Generer (new Evenement (Event_NEW_FRAME));

            break;
          }
```



```
        case Trame_ACK:
          {
            DEBUG ('l', "\t [D] Réception de l'ack %d (trame %d) \n",
                   trame->header.NR, trame->header.DebugInfo);

            // On signale l'événement au démon de réception
            evtEmission.
              Generer (new Evenement (Event_ACK_RECEIVED));
            break;
          }

        case Trame_REJECT:
          {
            ASSERT (FALSE);
            break;
          }

        case Trame_NUM: // [V5]
          {
            DEBUG ('l', "\t [D] Réception d'une trame NUM\n");
            // On signale l'évènement au démon d'émission (protocole)
            evtEmission.Generer (new Evenement (Event_NUM_RECEIVED));
            break;
          }

        default:
          {
            fprintf (stderr, "Oups: trame de type inconnu reçue !\n");
            interrupt->Halt ();
          }
        }                           // switch

    }                               // for
}
```