Projet réseau
Rapport

Thomas PROFETA
Sébastien DRANS

Table des matières

Introduction

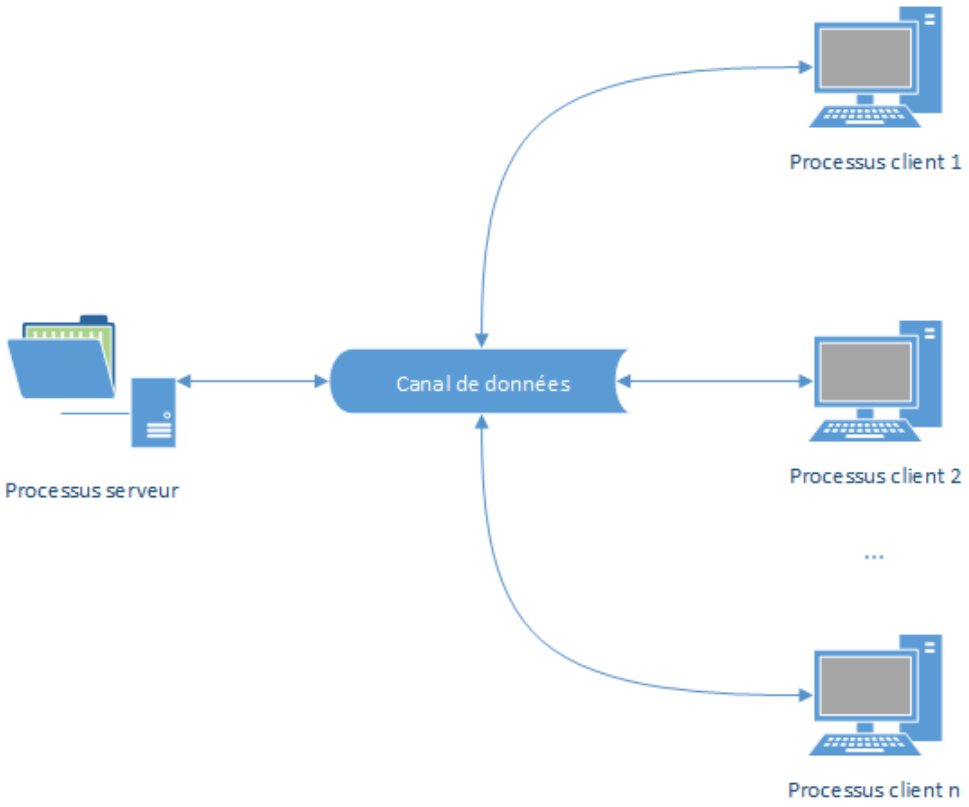
L'objectif de ce projet est de réaliser une étude, une conception et une implémentation des algorithmes usuellement déployés dans la couche session du modèle OSI pour permettre de gérer l'accès à une ressource partagée.

Pour cela nous nous inspirerons des algorithmes de la couche MAC du modèle TCP/IP (gestion d'accès au support) que l'on peut adapter et étendre pour répondre à cette problématique.

En effet, en généralisant :

- Les processus sont assimilables aux stations du réseau
- La ressource partagée joue le rôle du support

Ainsi, nous serons dans une configuration où un processus joue le rôle de serveur qui « héberge » un fichier, les autres jouant le rôle des clients qui souhaitent accéder à ce fichier pour y effectuer des opérations de lecture/écriture.



Opérations de bases

Les protocoles que nous définirons par la suite devront permettre de réaliser les opérations de base suivantes sur le fichier (ressource partagée).

Opération	Description
Lire (i)	Lit la ligne i du fichier (0 ≤ i < nbLigne)
Ecrire (i, str)	Ecrit la chaine str à la ligne i (0 ≤ i ≤ nbLigne) ; la ligne i devient la ligne i + 1
Compte ()	Compte le nombre de ligne du fichier

Algorithme inspiré de CSMA

Notes sur CSMA

CSMA (Carrier Sense Multiple Access) est un algorithme d'accès au support randomisé. Une station est libre d'émettre un message lorsqu'elle le souhaite. Pour cela, elle écoute au préalable le support de transmission, et, s'il est libre émet son message.

Bien que le principe général de cet algorithme de la couche MAC soit d'émettre un message sur le support lorsqu'il est libre, il existe 3 modes distincts pour déterminer si l'émission doit effectivement avoir lieu.

Modes d'émission

1-persitent

C'est le mode le plus simple ; lorsque la station veut émettre, elle commence à écouter le support, puis dès qu'il devient libre, elle émet son message.

P-persistent

Lorsque la station veut émettre, elle écoute le support, s'il est libre, elle émet son message avec une probabilité p, sinon attend un temps aléatoire puis réessaye.

O-persistent

Dans ce mode chaque station souhaitant émettre dispose d'un numéro d'ordonnancement fourni par une « station maître » constituant ainsi une file. Lorsque le support devient libre, la station de plus petit numéro (i.e. : en tête de file) peut émettre, les autres attendent qu'elle ait terminé (i.e. : le support est à nouveau disponible). Après chaque fin de transmission, les positions des stations sont mises à jour : « elles avancent dans la file ».

Extensions

CSMA/CD (Collision Detection)

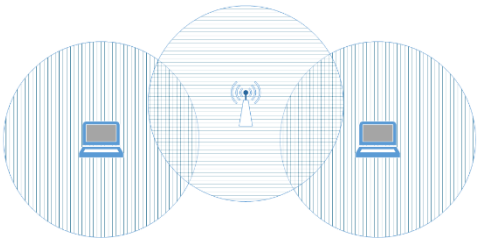
Dans CSMA, si deux stations détectent le support libre, elles émettront toutes deux en même temps : c'est une collision.

CSMA/CD répond à ce problème en donnant la possibilité aux stations de les détecter. Si une station détecte une collision, elle patientera pendant une durée définie aléatoirement puis tentera à nouveau d'émettre son message (dans la limite de 15 tentatives). Comme le temps d'attente est généré aléatoirement, il est très peu probable que les stations ayant provoqué une collision émettent à nouveau au même instant.

Remarque : CSMA/CD est l'algorithme d'accès au support utilisé dans Ethernet (802.3).

CSMA/CA (Collision Avoidance)

CSMA/CA est une adaptation de CSMA/CD aux réseaux sans fil. Dans CSMA/CD, les stations peuvent détecter les collisions car elles ont la possibilité de s'écouter. Or, dans le cas d'un réseau sans fil, il est possible que deux stations soient connectées au même point d'accès sans pouvoir le faire (station cachée).



Pour éviter les collisions, le point d'accès va avoir un rôle d'arbitrage. Si une station veut émettre, elle écoute le support ; s'il est libre, elle décide d'émettre avec une probabilité p (mode P-persistent). Si elle a décidé d'émettre, elle indique au point d'accès qu'elle est prête à le faire. Ce dernier, si personne n'est en train de transmettre des données, indique à la station qu'elle peut émettre (et aux autres qu'elle va le faire). Une fois les données transmises, le récepteur accuse réception.

Remarque : CSMA/CD est l'algorithme d'accès au support utilisé dans la norme Wi-Fi (802.11).

Adaptation

Nous nous baserons sur le principe de CSMA/CD (mode 1-persistant) pour créer l'algorithme d'accès à la ressource partagée.

Par conséquent, lorsqu'un processus client souhaitera accéder ou modifier la ressource partagée il vérifiera que personne n'accède à la ressource auprès du serveur. Si c'est le cas il transmettra ses données à ce dernier qui répondra à sa requête, sinon il patientera un temps aléatoire avant de réitérer sa demande.

Algorithme du processus client

```
1.  requêtes ← ensemble des requêtes à effectuer; //File des requêtes
2.  i ← 0;
3.  Tant que requêtes ≠ Ø faire
4.      Si i < 15 alors //On a essayé moins de 15 fois
5.          Si estLibre(serveur) alors //Le support est libre
6.              requêteATraiter ← défiler(requêtes);
7.              effectuer(requêteATraiter);
8.              i ← 0;
9.          Sinon //Le support n'est pas libre
10.             attendre un temps aléatoire;
11.             i ← i + 1;
12.         Fin si
13.     Sinon //On a essayé plus de 15 fois
14.         défiler(requêtes);
15.         erreur('Émission impossible : nombre d'essais dépassé');
16.         i ← 0;
17.     Fin si
18. Fin tant que
```

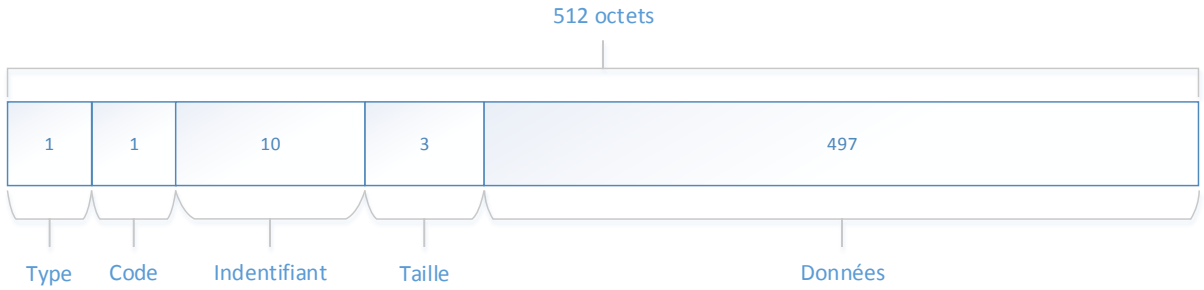
Algorithme du processus serveur

```
1.  Tant que VRAI faire
2.      requete ← prochaineRequete();
3.      Si requete ≠ Ø alors //Si un client a effectué une requête
4.          //Si un traitement n'est pas en cours
5.          Si traitementEnCours() = FAUX alors
6.              traiter(requete) ;
7.          Sinon //Si le serveur est occupé, on l'indique au client
8.              indiquer('erreur : serveur occupé', client(requete));
9.          Fin si
10.     Fin si
11. Fin tant que
```

Format des données échangées

Pour que le client et le serveur communiquent correctement il est nécessaire de définir un format d'échange des données.

Pour l'adaptation de l'algorithme CSMA/CD ce format est le suivant :



Description des champs

Type

Le champ « type » permet d'indiquer si la « trame » correspond à une requête ou à une réponse et prend les valeurs suivantes :

- Q pour une requête
- A pour une réponse

Code

Le champ « code » indique le code de l'opération :

- F pour une fonction
- E pour une erreur
- R pour un résultat

Identifiant

Le champ « identifiant » contient l'identifiant du couple requête/réponse entre un client et le serveur.

Taille

Le champ « taille » contient le nombre d'octets du champ « données ».

Remarque : la taille (en octet) des champs « type », « code », « identifiant » et « taille » est invariante.

Données

Le champ « données » contient une chaîne de caractères respectant le format suivant :

- Dans le cas d'une fonction
  - Nom fonction,argument 1,argument 2,…,argument n
    - Insertion d'un chaîne str à la ligne i : **wrt,i,str**
    - Lecture de la ligne i : **rd,i**
    - Compter le nombre de lignes du fichier : **cnt**
- Dans le cas d'un résultat
  - La valeur du résultat
- Dans le cas d'une erreur
  - Le code de l'erreur
    - Numéro de ligne indiqué en dehors des bornes : **ioob**
      - (Index out of bounds)
    - Fonction inexistante : **uf**
      - (Unknown function)
    - Les arguments ne correspondent pas à la fonction : **ba**
      - (Bad arguments)
    - Serveur occupé : **sb**
      - (Server busy)

Remarque : le séparateur des arguments de fonctions étant défini par le caractère « , » la chaîne de caractères « str » de la fonction d'écriture « wrt » ne doit pas la comporter.

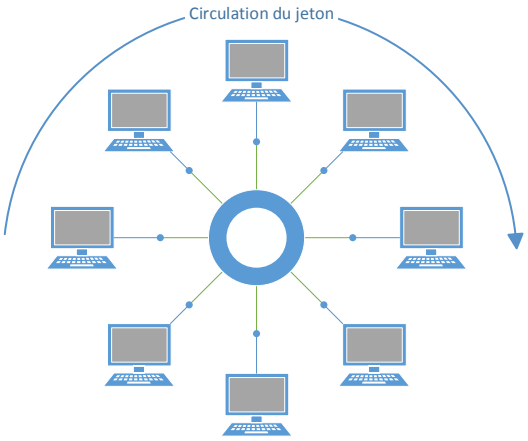
Récapitulatif

Champ	Description	Taille (en octets)	Valeurs
Type	Indique si la « trame » correspond à une requête ou une réponse	1	Q = requête A = réponse
Code	Indique le code de l'opération	1	F = fonction R = résultat E = erreur
Identifant	Identifiant d'un couple requête/réponse	10	Entier naturel
Taille	Nombre d'octets du champ « données »	3	Entier naturel
Données	Données de la « trame »	≤ 497	Chaine de caractères

Algorithme inspiré de Token Ring

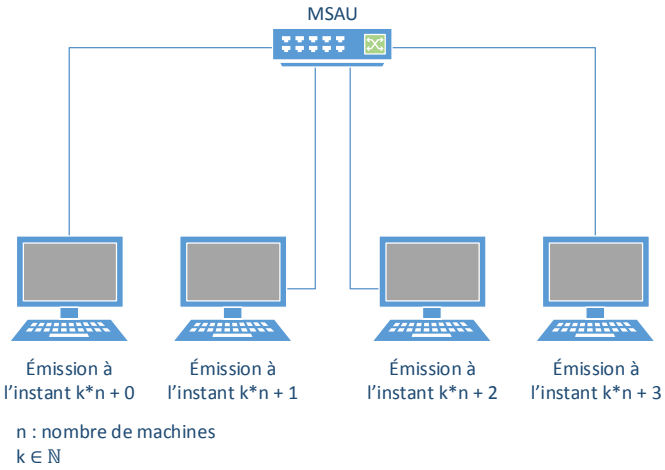
Notes sur Token Ring

Token Ring (anneau à jeton) est un algorithme d'accès au support se basant sur le principe du « droit de parole ». Dans ce protocole, les stations sont organisées en anneau. Un jeton symbolisant le droit d'émettre circule (toujours dans le même sens) de machine en machine au sein de l'anneau. Lorsqu'une machine reçoit le jeton, elle le conserve le temps d'émettre ses données, puis, le transmet à son successeur. Si elle n'a pas besoin d'émettre, elle le transmet directement.

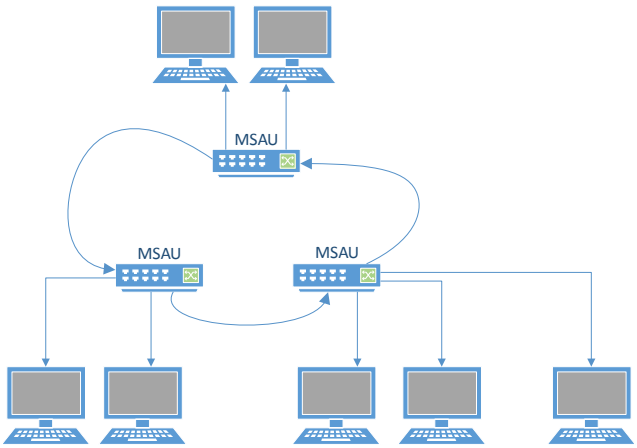


Remarque : une machine ne peut conserver le jeton que pendant une certaine durée pour éviter les monopoles. De plus, si une machine tombe en panne, elle est détectée et, le jeton ne lui est plus transmis pour ne pas bloquer le réseau.

Historiquement, le protocole Token Ring imposait une topologie physique en anneau puis, les MSAU (MultiStation Access Unit) ont permis d'implémenter ces réseaux avec une topologie en étoile. Pour cela, les machines sont organisées autour d'un nœud central qui est le MSAU. Dans un réseau Token Ring « classique » les machines transmettent directement le jeton à leur successeur. Ici, c'est le MSAU qui connaît l'ordonnancement des machines dans l'anneau et transmet le jeton : l'anneau est construit de manière logique plutôt que physique.



Un réseau Token Ring peut contenir plusieurs MSAUs. Dans ce cas, ces derniers sont organisés physiquement en anneau entre eux, les terminaux étant reliés aux MSAUs en adoptant une topologie en étoile.



Remarque : les MSAU peuvent remarquer qu'une machine est déconnectée est « sauter son tour » dans le passage du jeton pour ne pas bloquer le réseau.

Adaptation

Dans l'adaptation du protocole Token Ring, nous nous baserons sur son implémentation en étoile (l'anneau étant simulé grâce aux MSAUs).

Le processus serveur jouera le rôle du MSAU et distribuera le jeton aux processus client à tour de rôle. Lorsqu'un processus client reçoit le jeton, il peut s'il le souhaite transmettre des données, puis doit le retourner au processus serveur.

Algorithme du processus client

```
requêtes ← ensemble des requêtes à effectuer; //File des requêtes
Tant que requêtes ≠ ∅ faire
    Si disposeJeton() alors //On a le droit de parole
        requêteATraiter ← défiler(requêtes);
        effectuer(requêteATraiter);
        rendreJeton(); //On rend le droit de parole
    Sinon
        attendreJeton(); //On attend le droit de parole
    Fin si
Fin tant que
```

Algorithme du processus serveur

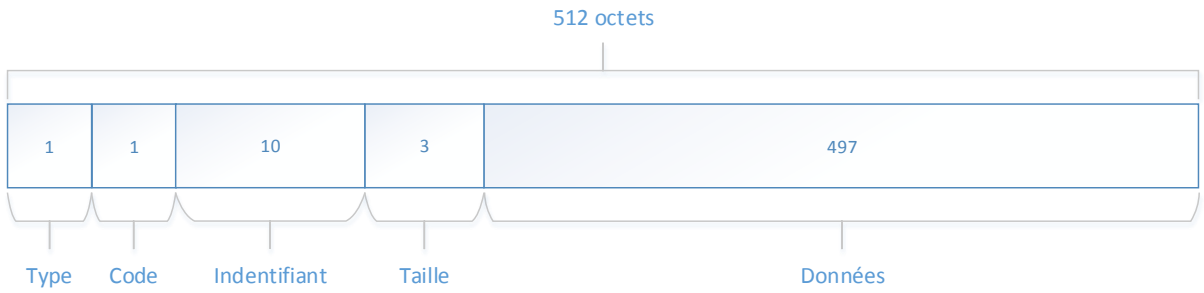
```
//Initialisation du jeton (chaque indice du tableau représente un
client, la valeur de la iᵉᵐᵉ case indique si la machine i dispose
du jeton)
disposeJeton[1..n];
disposeJeton[1..n] ← FAUX;
Tant que VRAI faire
    //Début de la ronde (une ronde correspond à un tour complet de
    l'anneau ; c'est-à-dire que chaque machine a eu la parole une
    fois)
    Pour i de 1 à n faire
        //Attribution du jeton à la machine d'indice i
        disposeJeton[i] ← VRAI;
        notifierDroitdeParole(i,durée);
        requete ← prochaineRequete();
        Si requete ≠ ∅ alors
            //La requête correspond à la machine qui a la parole
            Si client(requete) = i alors
                traiter(requete) ;
            Sinon
                indiquer('erreur : vous n'avez pas la parole',
                client(requete));
            Fin si
        Fin si
        //On retire le jeton à la machine d'indice i ; il sera
        transmis à son successeur dans la boucle suivante
        disposeJeton[i] ← FAUX;
        récupérerDroitDeParole(i); //Si le client garde le jeton
        plus de 2 fois le temps prévu, sa récupération est forcée
    Fin pour
Fin tant que
```

Remarque : le cas où une machine effectue une requête alors qu'elle ne dispose pas de la parole n'est pas censé avoir lieu si l'anneau fonctionne correctement, cependant nous le considérons tout de même pour des raisons de robustesse.

Format des données échangées

Le format d'une « trame » que nous utiliserons pour l'adaptation du protocole Token Ring est le même que celui utilisé pour CSMA/CD. Seules les fonctions et les codes d'erreurs utilisés seront modifiés pour permettre la passation du jeton.

Ainsi le format d'une « trame » pour notre algorithme Token Ring est le suivant :



Le type indique toujours si la « trame » est une requête (Q) ou une réponse (A). Nous utiliserons également le type de trame « commande » (C) qui sera utilisé pour la transmission du jeton. Le champ « identifiant » contient l'identifiant d'un couple requête/réponse sauf dans le cas d'un trame de type commande où sa valeur n'a pas de sens (il contiendra donc 0). Le code caractérise le type de l'opération (F pour une fonction, R pour un résultat et E pour une erreur). La taille indique le nombre d'octets contenus dans le champ « données ».

Fonctions

En plus des fonctions d'accès au fichier (wrt, rd, cnt) classiques qui restent inchangées, on utilise deux nouvelles fonctions qui permettent de gérer le jeton. Ces fonctions sont les suivantes :

- Give token : **gt,duree**
  - Cette fonction permet au serveur d'attribuer le jeton à un client pour une durée déterminée.
- Get back token : **gbt**
  - Fonction utilisée par le client pour rendre le jeton au serveur.

Erreurs

En plus des erreurs de base (ioob, uf, ba) on crée deux nouveaux codes d'erreurs :

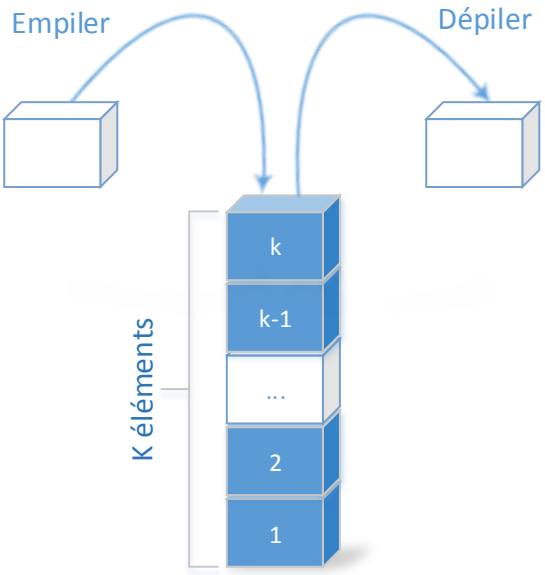
- Not your turn : **nyt**
  - Code d'erreur utilisé si un processus client essaye d'effectuer une requête alors qu'il ne dispose pas du jeton.
- Tocken has expired : **the**
  - Code d'erreur transmis à un client s'il conserve le jeton trop longtemps.

Algorithmes utilisant une pile

Notes sur le piles

Une pile est une structure de données basée sur le principe « dernier entré, premier sorti » (également appelé LIFO : « Last In First Out »).

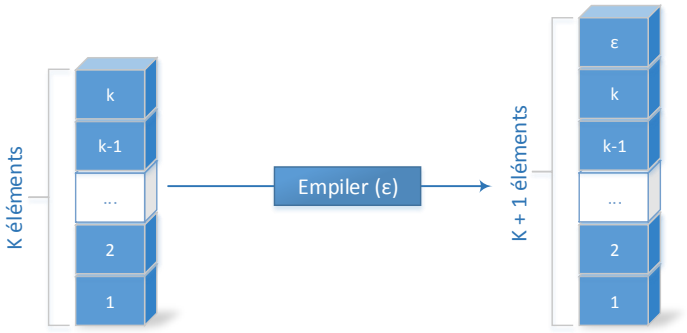
Remarque : la pile s'oppose à la file qui se fonde sur le principe « premier entré, premier sorti » (FIFO : « First In First Out »).



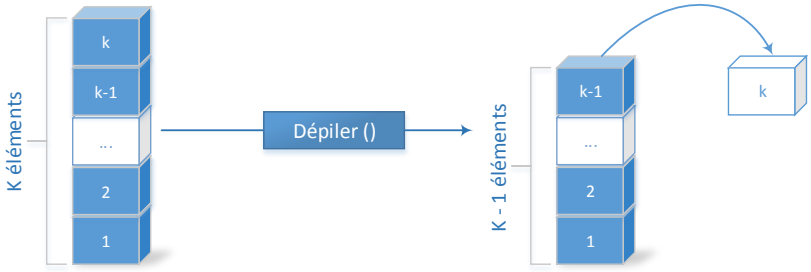
Pour simplifier, on peut voir une pile comme un empilement d'objets. Si l'on souhaite obtenir l'objet situé à la base de la pile il faut tout d'abord retirer tous les objets situés au-dessus. Ainsi, le dernier objet que l'on aura ajouté sera le premier que l'on devra retirer.

Une structure de données de type pile dispose de 3 opérations de base (primitives) qui sont les suivantes :

- Empiler (ε, π)
  - Permet d'ajouter l'élément ε à la pile π (si la pile π contenait k éléments, elle contient maintenant k + 1 éléments).



- Dépiler (π)
  - Retire l'élément au sommet de la pile π et le retourne (si la pile π contenait k éléments, elle en contient maintenant k – 1).



Remarque : on ne peut appeler la primitive dépiler () que si et seulement si la pile n'est pas vide.

- EstVide (π)
  - Permet de tester si la pile π contient des éléments ou non.

Algorithmes

Algorithme du processus client

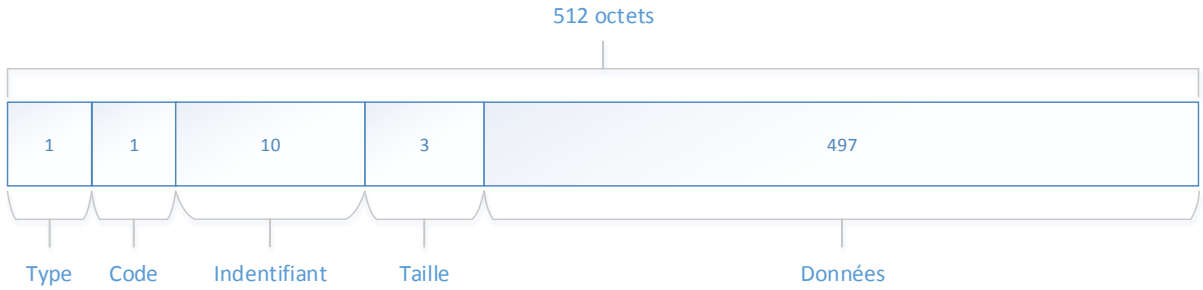
```
requêtes ← ensemble des requêtes à effectuer; //File des requêtes
Tant que requêtes ≠ ∅ faire //Tant qu'il existe des requêtes
    requêteATraiter ← défiler(requêtes);
    effectuer(requêteATraiter);
Fin tant que
```

Algorithme du processus serveur

```
π ← ∅; //Initialisation de la pile π
Tant que VRAI faire
    //Récupération de toutes les requêtes entrantes
    requetes ← prochainesRequetes();
    Si requetes ≠ ∅ alors //Si des requêtes existent
        //On les place successivement au sommet de la pile
        Pour chaque requete de la liste requetes faire
            empiler(requete,π);
        Fin pour
    Fin si
    Si estVide(π) = FAUX alors //Si la pile n'est pas vide
        //On récupère la requête situé à son sommet
        requeteATraiter ← depiler(π);
        traiter(requeteATraiter); //Puis on la traite
    Fin si
Fin tant que
```

Format des données échangées

Pour l'algorithme utilisant une pile nous continuerons d'utiliser le format de trame que nous avons défini précédemment :



Le type indique si la trame est de type requête (A) ou réponse (Q). Le code caractérise les données du champ « données » : il contiendra F pour fonction, R pour résultat, ou E pour erreur. L'identifiant permet d'associer les couples requête / réponse ensemble. Le champ taille indique le nombre d'octets contenu dans le champ « données ».

Fonctions et codes d'erreurs

Dans cet algorithme, nous n'avons pas besoin de fonctions spécifiques liées à son utilisation par conséquent ces dernières seront uniquement celles liées à la manipulation du fichier : « wrt », « cnt », et « rd ».

De même, aucun code d'erreur spécifique n'est nécessaire pour cet algorithme, par conséquent ces derniers seront limités aux codes d'erreur de base : « ioob », « uf », et « ba ».

Efficacité des algorithmes

Algorithme adapté de CSMA/CD

Dans notre adaptation de l'algorithme de CSMA/CD une station qui souhaite émettre écoute le support et, s'il est libre émet. Sinon, elle attend une durée aléatoire avant de réessayer (en « espérant » que le support sera libre). Après 15 tentatives infructueuses, elle considère que l'émission de la trame est impossible.

Par conséquent, cet algorithme est dit « probabiliste » : il ne permet pas de garantir qu'une trame sera délivrée. Cependant la probabilité qu'une trame ne soit pas émise reste très faible.

Le mode de fonctionnement « probabiliste » permet à chaque station de tenter d'accéder au serveur à tout moment : il n'y a pas d'intervalle de temps inutilisés (c.f. : adaptation de Token Ring) ce qui accroît l'efficacité de cet algorithme.

Algorithme adapté de Token Ring

L'algorithme adapté de Token Ring « découpe » le temps en intervalles discrets. À chaque intervalle une station disposera du droit de parole et pourra transmettre des données si elle le désire. Contrairement à l'adaptation de CSMA/CD, cet algorithme est « déterministe » : il garantit que chaque trame sera délivrée.

Mais, comme à chaque intervalle de temps cet algorithme interroge une station pour savoir si elle souhaite émettre des données, il perd en efficacité. En effet, supposons qu'une seule station souhaite émettre lors d'une ronde, l'algorithme va interroger les n stations de l'anneau pour qu'une seule transmettre des données. Au final nous avons eu besoin de n intervalles pour transmettre un message.

Algorithme utilisant les piles

L'algorithme utilisant les piles traite les requêtes dans l'ordre inverse de leur arrivée : les dernières arrivées sont les premières traitées. Par conséquent, si le processus serveur reçoit de nombreuses requêtes, il y a un risque de famine.

En effet supposons qu'un processus client (appelons-le Maurice) envoie une requête au serveur, puis, que d'autres processus envoient également des requêtes. La requête de Maurice est donc à la base de la pile. Le serveur va prendre en charge la requête des autres processus en priorité. Si, avant que le serveur ne traite la requête de Maurice de nouvelles requêtes arrivent, elles seront à nouveau prioritaires, et Maurice continuera d'attendre sa réponse pendant le traitement des autres requêtes. Maurice n'obtiendra sa réponse que si toute la pile des requêtes finit par être dépilée, c'est-à-dire que les autres processus n'auront pas fait de demandes pendant un certain temps.

Récapitulatif

Algorithme	Performance	Déterministe	Équitable
CSMA/CD	ooooo	✗	✓
Token Ring	oo	✓	✓
Pile	oooo	✓	✗