

Le Distributed Computing Environment de OSF

- [Définition](#)
 - [L'architecture de DCE](#)
 - [L'organisation en cellules](#)
 - [Comment former une cellule](#)
 - [Les RPC sous DCE](#)
 - [Le "stub" client](#)
 - [Le RPCRuntime](#)
 - [Le "stub" serveur](#)
 - [Génération des stubs](#)
 - [Distributed File System](#)
 - [L'accès aux fichiers DFS](#)
 - [La gestion de la cohérence](#)
-

Définition

DCE = Distributed Computing Environment, de l'OSF (Open Software Foundation).

OSF est un consortium de fabricants d'ordinateurs (IBM, DEC, HP, ...).

DCE n'est PAS un OS. C'est un ensemble de services et d'outils, qui tournent sur un OS existant, qui servent à la création et au déroulement d'applications distribuées.

DCE est indépendant des machines et des OS. On peut l'utiliser sur AIX, SunOS, Unix System V, Windows, OS/2, ...

DCE supporte aussi de nombreux matériels et logiciels réseaux (TCP/IP, X.25, ...).

L'approche DCE est l'inverse de l'approche micro-kernel.

Historique : DCE n'a pas été écrit "from scratch" (à partir de rien), il a été conçu à partir d'un "call for technology", pour obtenir les meilleures solutions aux problèmes de distribution.

L'architecture de DCE

Il y a 6 composants :

1. *Threads package*, la gestion des threads ;
2. *Remote Procedure Call facility*, la gestion des RPCs et du paradigme client/serveur ;
3. *Distributed Time Service*, la notion de temps global ;
4. *Name services*, la gestion des noms :
 - Cell Directory Service,
 - Global Directory Service,
 - Global Directory Agent ;
5. *Security Service*, la gestion de la sécurité (authentification et autorisation) ;
6. *Distributed File Service*, le système distribué de gestion de fichiers.

L'organisation en cellules

DCE est un système qui peut être fortement étendu (il est "highly scalable").

On peut rajouter des machines et des utilisateurs sans (trop) nuire aux performances.

Organisation en **cellules**, qui sont des unités manageable de taille raisonnable.

Une cellule = un ensemble d'utilisateurs, de machines ou autres qui ont un but en commun et partagent des services DCE communs.

Chaque cellule comprend au minimum : un serveur de répertoires de cellules, un serveur de sécurité et un serveur de temps global + des machines clientes.

Chaque client DCE a des processus clients pour gérer les facilités DCE.

Comment former une cellule

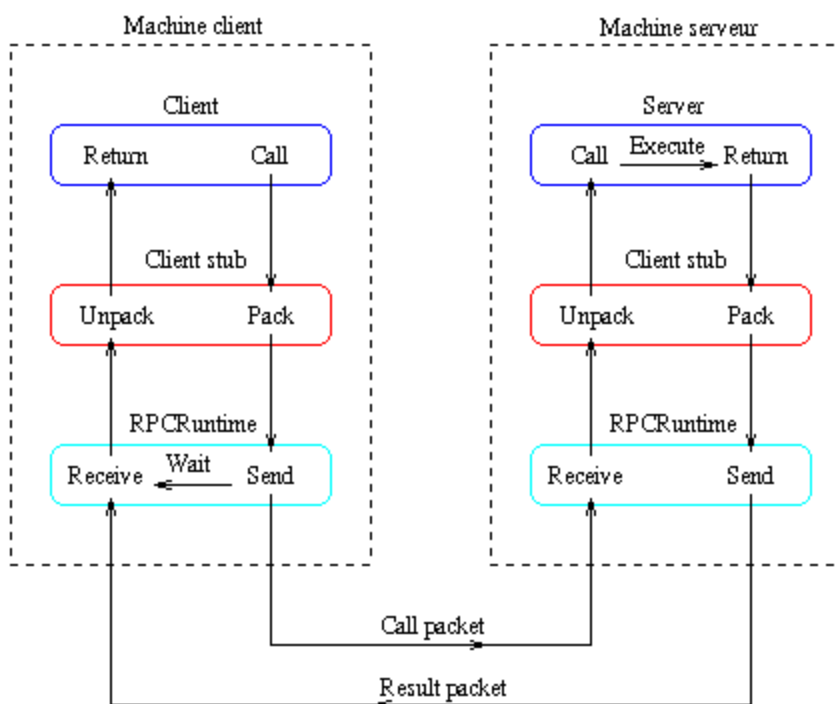
- Par *but commun*, les personnes travaillant à un même but gagnent à être regroupées dans la même cellule.
- Par *intérêt administratif*, il est plus facile d'administrer des utilisateurs si ceux-ci sont regroupés dans une seule cellule.
- Par *souci de sécurité*, on préférera mettre dans une même cellule les machines d'utilisateurs qui ont le même degré de fiabilité.
- Par *coût d'utilisation*, les usagers qui interagissent fortement entre eux seront placés dans une même cellule.

Les RPC sous DCE

RPC = la base de toute communication dans DCE.

Les RPC-DCE dérivent du Network Computing System (NCS) développé par Apollo (partie de HP).

LES RPC-DCE utilisent la génération automatique de **stubs** :



Le "stub" client

Deux tâches :

1. CALL REQUEST.
 - reçoit un "call request" du client,
 - forme un message (pack) avec les spécifications de la procédure distante et les paramètres,
 - demande au RPCRuntime de les transmettre au stub serveur.
2. RESULT.
 - reçoit le résultat de l'exécution de la procédure du RPCRuntime,
 - décode le message (unpack),
 - transmet les données au client.

Le RPCRuntime

Gère la transmission des messages entre le client et le serveur.

Responsable des retransmissions, acknowledges, etc.

Reçoit les messages du stub client et les envoie au stub serveur.

Reçoit les résultats du stub serveur et les envoie au stub client.

Le "stub" serveur

Deux tâches :

1. CALL REQUEST.
 - reçoit un "call request" du RPCRuntime,
 - décode le message (unpack),
 - exécute l'appel de procédure locale (normalement) dans le serveur.
2. RESULT.
 - reçoit le résultat de l'exécution de la procédure du serveur,
 - forme un message (pack) avec les résultats,
 - transmet les données au RPCRuntime.

Génération des stubs

Les stubs peuvent être générés de 2 façons :

1. *manuellement*, le concepteur des RPC offre un ensemble de fonctions pour que l'utilisateur fabrique ses propres stubs. Avantage : simple à implémenter (pour le concepteur) et peut gérer des paramètres de type complexe.
2. *automatiquement*, dans ce cas il faut utiliser un IDL (Interface Description Language) pour définir l'interface entre le client et le serveur. Cette interface est une liste de procédure, avec le type des paramètres et des résultats.

Exemple d'une interface écrite avec un IDL :

```
[uuid (bf sdfw345-345-3245-qwef-356-we45-ew54w-e4-5w-345)
 version (1.0)]
```

```
interface stateless_fs
{
    const long FILE_NAME_SIZE = 16
    const long BUFFER_SIZE = 1024
    typedef char FileName[FILE_NAME_SIZE];
    typedef char Buffer[BUFFER_SIZE];
```

```

void read (
    [in] FileName    filename;
    [in] long        position;
    [in,out] long    nbytes;
    [out] Buffer      buffer;
);

void write (
    [in] FileName    filename;
    [in] long        position;
    [in,out] long    nbytes;
    [in] Buffer      buffer;
);
}

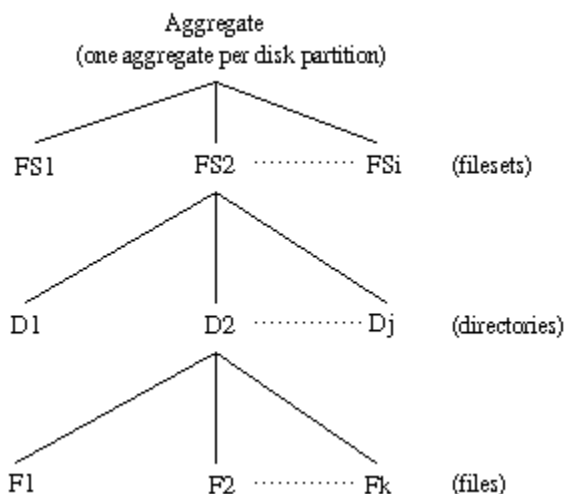
```

Distributed File System

DFS est un système distribué de gestion de fichiers.

Comme sous Unix, les fichiers ne sont pas structurés, et sont vus comme une suite de bytes.

DFS possède 4 niveaux d'agrégation :



Un fileset est un groupe de fichier semblable à un *file system* Unix.

Pourtant, à la différence d'Unix, un fileset est un sous-ensemble d'un file system (plusieurs filesets dans un aggregate).

Facilite la gestion des fichiers (un fileset = tous les fichiers d'un utilisateur ou d'un groupe d'utilisateurs, ...).

Lorsqu'une partition devient pleine, on peut dynamiquement faire migrer un fileset de cette partition vers une autre (avec plus d'espace).

L'accès aux fichiers DFS

Paradigme client / serveur et utilisation d'un cache de données (comme NFS).

Une machine peut être soit client, soit serveur, soit les deux.

Du côté du serveur, on trouve :

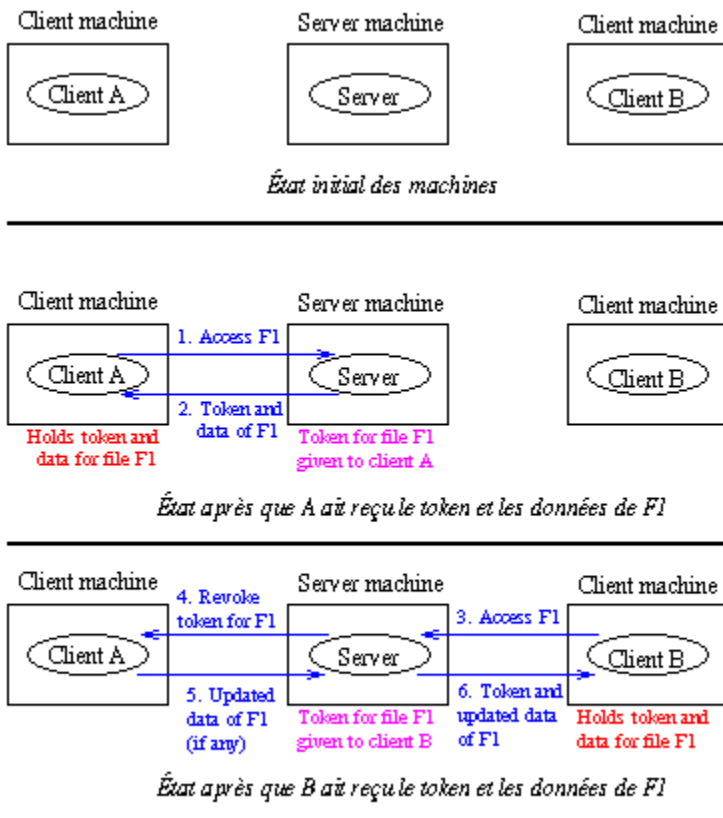
- Au niveau du Kernel :
 1. *Episode*, c'est le file system local ;
 2. *Token manager*, les jetons sont utilisés pour gérer les problèmes de cohérence du cache ;
 3. *File exporter*, accepte les requêtes des clients et leur répond. Les interactions se font grâce aux RPC. Accepte aussi les demandes d'authentification pour l'établissement de connexions sûres.
- Au niveau de l'espace utilisateur :
 1. *Fileset server*, gère les filesets locaux ;
 2. *Fileset location server*, conserve les informations sur quel DFS server gère quels filesets. ;
 3. *Replication server*, maintient la consistance des filesets répliqués.

Note 1 : la taille d'une unité de transfert est de 64 Ko, à comparer avec les 8 Ko de NFS.

Note 2 : DFS n'a pas de mécanisme de *read-ahead*.

La gestion de la cohérence

Caractéristique principale de DFS : chaque opération "read" voit les effets des précédentes opérations write.



Il existe différents types de token, suivant l'opération à effectuer :

1. *type-specific tokens*, il existe des tokens spécifiques pour les opérations : read, write, open, lock, check, ...
2. *Fine-grained tokens*, pour minimiser le false sharing, les tokens (pour read, write ou lock) ne concernent qu'une **partie** du fichier.

Chaque token a un délai d'expiration de 2 mn.



[Retour au sommaire.](#)