

Amoeba

- [Définition](#)
 - [Architecture matérielle](#)
 - [Architecture logicielle](#)
 - [Fonctionnalités du micro-kernel](#)
 - [Serveurs](#)
 - [Objets](#)
 - [Serveur de fichiers](#)
 - [Objets et capacités](#)
 - [Protection des objets](#)
 - [Opérations standards sur les objets](#)
 - [Gestion des processus](#)
 - [Exécution des processus](#)
-

Définition

Amoeba = Système d'exploitation distribué.

Projet commencé en 1981 à l'Université de Vrije (Pays-Bas).

But du projet = OS distribué pour le calcul parallèle (plusieurs processeurs) et distribué.

L'utilisateur se logge, édite des programme, les compile, lit son mail, etc.

Ces actions font intervenir différentes machines.

L'utilisateur ne le voit pas.

Un utilisateur ne se logge pas sur **une machine particulière**, mais au **système entier**.

Il n'y a pas de "home machine".

Le shell s'exécute sur une machine quelconque.

Les commandes s'exécutent sur des machines quelconques (en général différente de celle du shell).

⇒ le système recherche à chaque fois la machine la moins chargée.

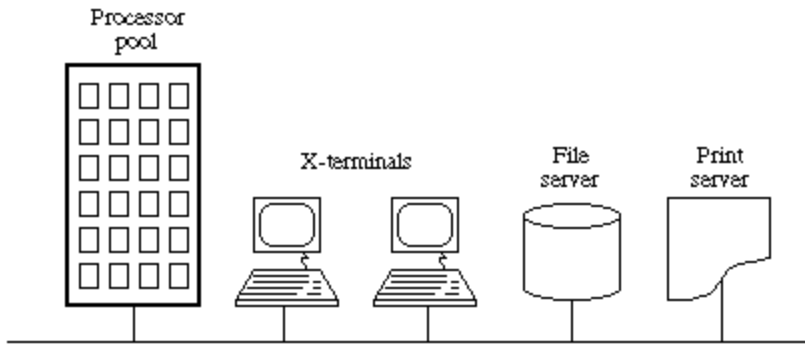
⇒ Amoeba est très "location transparent".

Exemple de commande : `amake` (amoeba's make).

C'est le système qui détermine les compilations qui peuvent s'exécuter en parallèle ou en série et sur quelle(s) machine(s).

Un langage **spécifique** qui tient compte du parallélisme et de la distribution a été créé : **Orca**.

Architecture matérielle



1. Systèmes avec nombreux CPUs.
2. Chaque CPU mémoire de dizaines de Mo.

Pari : plus rentable d'avoir quelques systèmes multi-processeurs que des stations individuelles.

- Utilisation maximum des processeurs (pas de stations dormantes dans les placards, cf. "big router is watching you").
- Les CPUs dans un pool peuvent être de natures différentes. Théoriquement, le fils d'un processus peut tourner sur un processeur différent que son père.

Un pool de processeurs est moins cher qu'un ensemble de stations.

Ce sont des cartes avec une connexion réseau.

Pas d'écran, de clavier ou de souris et partage de la même source d'énergie.

Si on a pas de pool de processeurs, on peut utiliser un pool de stations, éventuellement délocalisées.

On accède à Amoeba à travers un terminal X banalisé.

Architecture logicielle

2 parties distinctes :

1. un micro-noyau (1 par processeur) ;
2. un ensemble de serveurs qui implémentent les opérations classiques d'un SE.

Fonctionnalités du micro-kernel :

1. gestion des processus et threads ;
2. gestion de base de la mémoire ;
3. gestion des communications ;
4. gestion des E/S de base.

Fonctionnalités du micro-kernel

Notion de processus : x processus = x espaces d'adressage distincts.

Notion de threads : x threads dans 1 processus = 1 seul espace d'adressage.

Gestion basique de la mémoire : les threads peuvent allouer et libérer des segments de mémoire.

Gestion des communications : paradigme client / serveur et diffusion. Le protocole de communication est FLIP.

Gestion des E/S : pour chaque périphérique est associé un pilote (driver) dans le noyau.

Serveurs

Tout ce qui n'est pas fait par le kernel est fait par des **serveurs**.

⇒ Minimiser la **taille** du kernel.

⇒ Accroître la **flexibilité** : changer un serveur ou une version de serveur est facile.

On peut avoir en même temps plusieurs versions différentes pour des utilisateurs différents.

Au coeur de Amoeba = notion d' **objet**.

Objets

Objet = Abstract Data Type pour Amoeba.

Données encapsulées + Opérations.

Un processus crée un objet ⇒ le serveur qui gère cela retourne une capacité (**capability**) protégée par **cryptage**.

Tous les objets du système (objets physiques ou logiques) sont : nommés, protégés et gérés par des capacités.

Exemple : fichiers, directories, segments mémoire, fenêtres, processeurs, disques, etc.

Interface uniforme ⇒ généricité et simplicité.

Serveur de fichiers

C'est le **bullet server** (bullet car il doit être rapide).

Une fois créé, un fichier **ne peut pas être modifié**.

⇒ Moins de problèmes de cohérence des données.

⇒ Duplication facilitée.

Il y a le **directory server** qui gère les directories et le **replication server**.

L'utilisateur est libre d'utiliser les serveurs d'origine ou de créer ses propres serveurs.

Objets et capabilities

Une opération sur un objet = RPC sur le serveur qui le gère.

Adresse des objets non-importante (même machine ou à l'autre bout du pays, ...).

Capacité =



Un client veut faire une opération :

- Appel d'une stub-procedure qui construit un message contenant la capability ; passe la main au kernel.
- Le kernel extrait le **server port**, regarde son cache pour savoir où le serveur se trouve.
- Si le port n'est pas dans le cache → broadcast "où est-il ?".
- Le port est une adresse logique.
- La capability est envoyée au serveur.
- Le champ **object** est utilisé par le serveur pour localiser l'objet.

Protection des objets

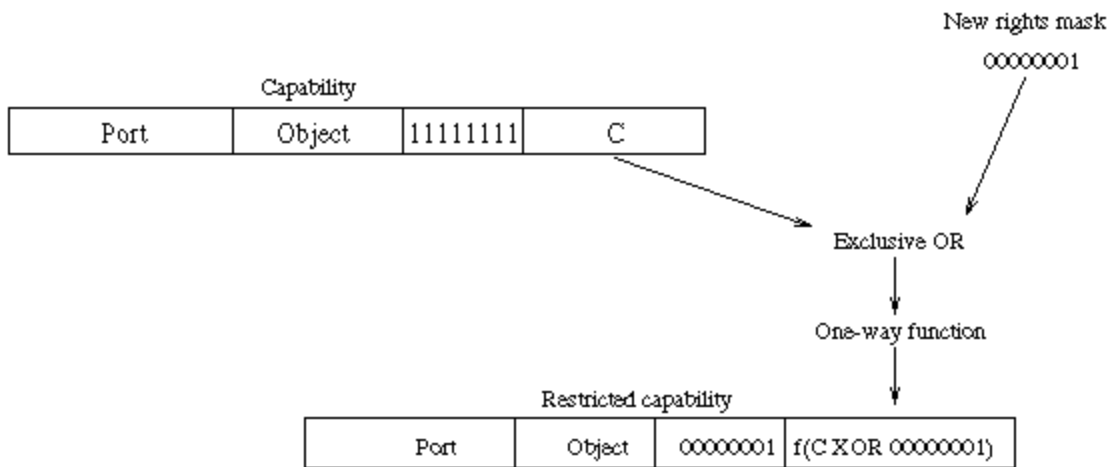
Quand un objet est créé, le champ **Check** est choisi au hasard est stocké dans la capability et dans les tables.

La capability est renvoyé au client avec tous les droits ON.

Si le client veut utiliser lui-seul l'objet, il restreint les droits.

Quand le serveur reçoit une capability → XOR avec les nouveaux droits → fonction one-way.

$y = f(x)$ avec x facile de trouver y . Avec y , parcours exhaustif de toutes les valeurs possibles de x .



Opérations standards sur les objets

- Age → garbage collection ;
- Copy → copie l'objet et renvoie une capability ;
- Destroy → détruit l'objet, libère la place ;
- Getparams → récupère les params du serveur ;
- Info → chaîne ASCII décrivant l'objet ;
- Restrict → produit une nouvelle capacité avec de nouveaux droits ;
- Setparams → envoie de nouveaux paramètres ;
- Status → renvoie le status du serveur ;
- Touch → fait comme si l'objet vient juste d'être utilisé.

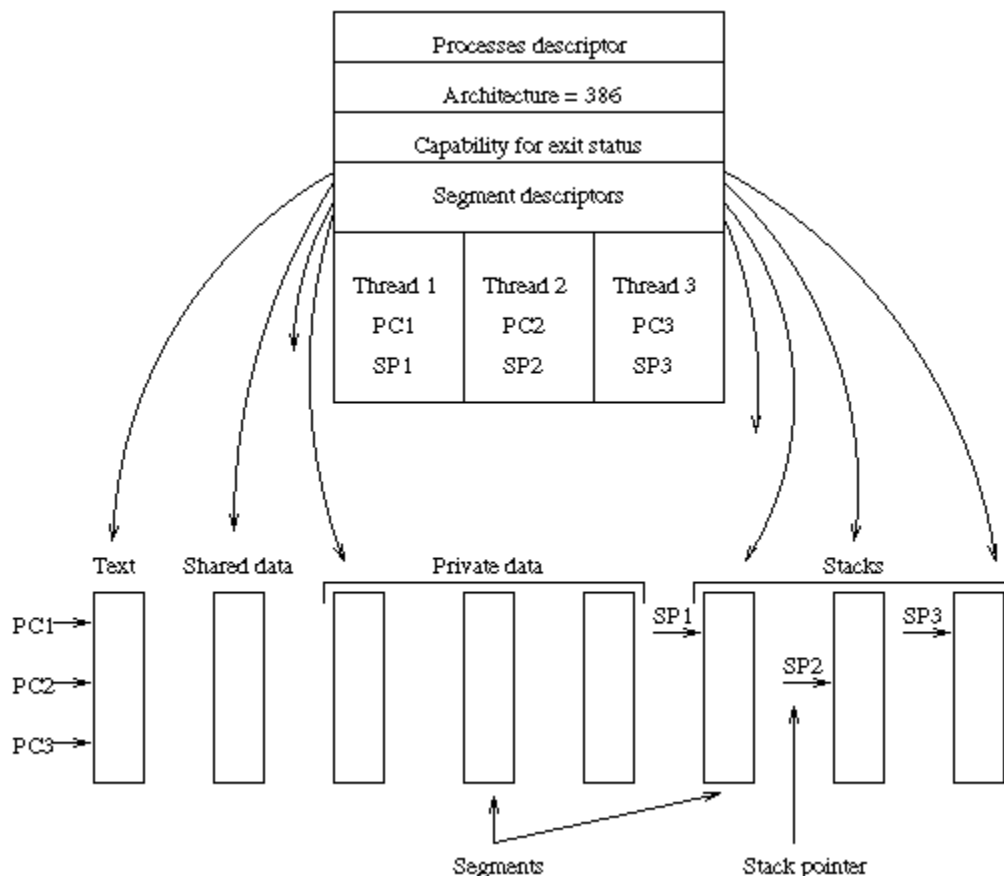
Gestion des processus

1 processus est un objet.

Différent de Unix (pas de duplication du processus père).

Process descriptor : contient par exemple :

- sur quel(s) processeur(s) le processus peut être exécuté ;
- la capability ;
- la description des segments mémoire ;
- la description des threads.



Exécution des processus

Une primitive de bas niveau : `exec`.

Prend en paramètre le descripteur de processus + capability du serveur.



[Retour au sommaire.](#)