La Programmation informatique

La **programmation** dans le domaine informatique est l'ensemble des activités qui permettent l'écriture des programmes informatiques. C'est une étape importante de la conception de logiciel (voire de matériel, cf. VHDL).

Pour écrire le résultat de cette activité, on utilise un langage de programmation.

La programmation représente usuellement le codage, cest-à-dire la rédaction du code source d'un logiciel. On utilise plutôt le terme développement pour dénoter l'ensemble des activités lié à la création d'un logiciel.

Une brève histoire de la programmation

La première machine programmable (cest-à-dire machine dont les possibilités changent quand on modifie son "programme") est probablement le métier à tisser de Jacquard, qui a été réalisé en 1801. La machine utilisait une suite de cartons perforés. Les trous indiquaient le motif que le métier suivait pour réaliser un tissage ; avec des cartes différentes le métier produisait des tissages différents. Cette innovation a été ensuite améliorée par Herman Hollerith d'IBM pour le développement de la fameuse carte perforée d'IBM.

En 1936, la publication de l'article fondateur de la science informatique On Computable Numbers with an Application to the Entscheidungsproblem[1] par Alan Mathison Turing allait donner le coup d'envoi à la création de l'ordinateur programmable. Il y présente sa machine de Turing, le premier calculateur universel programmable, et invente les concepts et les termes de programmation et de programme.

Les premiers programmes d'ordinateurs étaient réalisés avec un fer à souder et un grand nombre de tubes à vide (plus tard, des transistors). Les programmes devenant plus complexes, cela est devenu presque impossible, parce qu'une seule erreur rendait le programme entier inutilisable. Avec les progrès des supports de données, il devient possible de charger le programme à partir de cartes perforées, contenant la liste des instructions en code binaire spécifique à un type d'ordinateur particulier. La puissance des ordinateurs augmentant, on les utilisa pour faire les programmes, les programmeurs préférant naturellement rédiger du texte plutôt que des suites de 0 et de 1, à charge pour l'ordinateur d'en faire la traduction lui-même. Avec le temps, de nouveaux langages de programmation sont apparus, faisant de plus en plus abstraction du matériel sur lequel devaient tourner les programmes. Ceci apporte plusieurs facteurs de gains : ces langages sont plus faciles à apprendre, un programmeur peut produire du code plus rapidement, et les programmes produits peuvent tourner sur différents types de machines.

La fin des programmeurs ?

De tous temps, on a prédit « la fin des programmeurs ».

Dans les années 60, les langages symboliques comme AUTO-CODE, Cobol et Fortran ont en effet mis fin - en grande partie - à la programmation de bas niveau tel que l'assembleur. Il semblait alors clair que n'importe qui était capable d'écrire du code du type

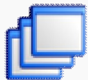
```
multiply MONTANT-HT by TAUX-TVA giving MONTANT-TAXES.
add MONTANT-HT, MONTANT-TAXES giving MONTANT-TTC.
```

ou

```
RDELTA = SQRT(B**2 - 4*A*C)
X1 = (-B + RDELTA) / (2*A)
```

plutôt que des dizaines de lignes cryptiques comme

```
movl    %esp, %ebp
subl    $24, %esp
flds    12(%ebp)
fmuls   12(%ebp)
flds    8(%ebp)
flds    .LC0
fmulp   %st, %st(1)
fmuls   16(%ebp)
fsubrp  %st, %st(1)
fstpl   (%esp)
call    sqrt
fstps   -4(%ebp)
flds    -4(%ebp)
fsubs   12(%ebp)
flds    8(%ebp)
fadd    %st(0), %st
fdivrp  %st, %st(1)
```



Il existe une catégorie dédiée à ce sujet : **Méthode de développement logiciel.**

Pourtant il a vite fallu se rendre compte que la programmation ne se limitait pas au codage, et que la conception d'applications était un vrai métier qui ne s'improvise pas.

Dans les années 80, la micro-informatique a souvent conduit à une informatisation sauvage des entreprises, dont le service informatique débordé n'arrivait pas à satisfaire les demandes (qui ne correspondaient d'ailleurs pas forcément aux vrais besoins des utilisateurs finaux). D'où la réalisation d'applications bricolées par des stagiaires « petit génie » en Basic et autres macros de tableurs, répondant à un besoin ponctuel, mais créant de fait un système d'information parallèle non maintenu. Dans cette catégorie, on peut également partiellement ranger les applications développées avec

des L4G par des utilisateurs non formés. Il est évident que ce type d'outil (dont MS Access est un descendant) permet à un utilisateur non formé de réaliser de petites applications qu'il n'aurait pas entreprises autrement, et à un utilisateur compétent de développer très rapidement d'importantes applications, mais elles n'ont pas le pouvoir magique de faire l'analyse et la conception automatiquement dans les mains d'un utilisateur novice.

Phases de création d'un programme

Conception [modifier]

Articles détaillés : Conception de logiciel et Algorithmique.

La phase de conception définit le but du programme. Si on fait une rapide analyse fonctionnelle d'un programme, on détermine essentiellement les données qu'il va traiter (données d'entrée), la méthode employée (appelée l'algorithme), et le résultat (données de sortie). Les données d'entrée et de sortie peuvent être de nature très diverses. On peut décrire la méthode employée pour accomplir le but d'un programme à l'aide d'un algorithme. La programmation procédurale et fonctionnelle est basée sur l'algorithmique. On retrouve en général les mêmes fonctionnalités de base :

Pour la programmation impérative

Article détaillé : Programmation impérative.

"Si"

Si *prédicat*

Alors *faire ceci*

Sinon *faire cela*

"Tant que"

Tant que *prédicat*

Faire ...

"Pour"

Pour *variable* **allant de** *borne inférieur* **à** *borne supérieur*

Faire ...

Codage

Article détaillé : Langage de programmation.

Une fois l'algorithme défini, l'étape suivante est de coder le programme. Le codage dépend de l'architecture sur laquelle va s'exécuter le programme, de compromis temps-mémoire, et d'autres contraintes. Ces contraintes vont déterminer quel langage de programmation utiliser pour "convertir" l'algorithme en code source.

Transformation du code source

Le code source n'est (presque) jamais utilisable tel quel. Il est généralement écrit dans un langage "de haut niveau", compréhensible pour l'homme, mais pas pour la machine.

Compilation

Articles détaillés : Compilation (informatique) et Machine virtuelle.

Certains langages sont ce qu'on appelle des langages compilés. En toute généralité, la compilation est l'opération qui consiste à transformer un langage source en un langage cible. Dans le cas d'un programme, le compilateur va transformer tout le texte représentant le code source du programme, en code compréhensible pour la machine, appelé code machine.

Dans le cas de langages dits compilés, ce qui est exécuté est le résultat de la compilation. Une fois effectuée, l'exécutable obtenu peut être utilisé sans le code source.

Il faut également noter que le résultat de la compilation n'est pas forcément du code machine correspondant à la machine réelle, mais peut être du code compris par une machine virtuelle (c'est à dire un programme simulant une machine), auquel cas on parlera de bytecode. C'est par exemple le cas en Java. L'avantage est que, de cette façon, un programme peut fonctionner sur n'importe quelle machine réelle, du moment que la machine virtuelle existe pour celle-ci.

Dans le cas d'une requête SQL, la requête est compilée en une expression utilisant les opérateurs de l'algèbre relationnelle. C'est cette expression qui est évaluée par le système de gestion de bases de données.

Interprétation

Article détaillé : Interpretation (informatique).

D'autres langages ne nécessitent pas de phase spéciale de compilation. La méthode employée pour exécuter le programme est alors différente. Le programme entier n'est jamais compilé. Chaque ligne de code est compilée "en temps réel" par un programme. On dit de ce programme qu'il interprète le code source. Par exemple, python est un langage interprété.

Cependant, ce serait faux de dire que la compilation n'intervient pas. L'interprète produit le code machine, au fur et à mesure de l'exécution du programme, en compilant chaque ligne du code source.

Avantages, inconvénients

Les avantages généralement retenus pour l'utilisation de langages "compilés", est qu'ils sont plus rapides à l'exécution que des langages interprétés, car l'interprète doit être lancé à chaque exécution du programme, ce qui mobilise systématiquement les ressources.

Traditionnellement, les langages interprétés offrent en revanche une certaine portabilité (la capacité à utiliser le code source sur différentes plate-formes), ainsi qu'une facilité pour l'écriture du code. En effet, il n'est pas nécessaire de passer par la phase de compilation pour tester le code source.

Appellation impropre

Il faut noter qu'on parle abusivement de langages compilés ou interprétés. En effet, le caractère compilé ou interprété ne dépend pas du langage, qui n'est finalement qu'une grammaire et une certaine sémantique. D'ailleurs, certains langages, comme ruby, peuvent être utilisés interprétés ou compilés.

Néanmoins, l'usage qu'on fait des langages est généralement fixé.

Test du programme

Article détaillé : Test (informatique).

C'est l'une des étapes les plus importantes de la création d'un programme. En principe, tout programmeur se doit de vérifier chaque partie d'un programme, de le tester. Il existe différents types de test. On peut citer en particulier:

- Test unitaire
- Test d'intégration
- Test de performance

Il convient de noter qu'il est parfois possible de vérifier un programme informatique, c'est à dire prouver, de manière plus ou moins automatique, qu'il assure certaines propriétés.

Pratiques

- Algorithmique
- Gestion de versions
- Optimisation du code
- Programmation système
- Refactoring
- Test d'intégration

- Test unitaire

Techniques de programmation

- Programmation concurrente
- Programmation déclarative
- Programmation fonctionnelle
- Programmation impérative
- Programmation logique
- Programmation orientée aspect
- Programmation orientée composant
- Programmation orientée objet
- Programmation orientée prototype
- Programmation par contraintes
- Programmation par contrat
- Programmation par intention
- Programmation procédurale
- Programmation structurée

COBOL

COBOL est un langage de programmation de troisième génération créé en 1959 (officiellement le 18 Septembre 1959[1]). Son nom est l'acronyme de **COmmon Business Oriented Language** qui révèle sa vocation originelle : être un langage commun pour la programmation d'applications de gestion. Le langage COBOL était de loin le langage le plus employé des années 1960 à 1980, et reste utilisé dans des grandes entreprises, notamment dans les institutions financières qui disposent de nombreux logiciels en COBOL.

Préhistoire et spécifications

Le COBOL a initialement été créé en 1959 par le Short Range Committee, un des trois comités proposés à une rencontre au Pentagone en mai 1959 organisée par Charles Phillips du département de la défense des États-Unis. Le comité a été formé pour recommander une approche à court terme pour un langage commun, indépendant des constructeurs, pour les applications de gestion de l'administration américaine. Il était constitué de membres représentant six constructeurs d'ordinateurs et trois agences gouvernementales. Les six constructeurs informatiques étaient Burroughs Corporation, IBM, Minneapolis-Honeywell, RCA, Sperry Rand, et Sylvania Electric Products. Les trois agences du gouvernement étaient le US Air Force, le David Taylor Model Basin, et l'Institut national des standards. Ce comité était présidé par un membre du NBS. Des comités à moyen et long terme ont également été proposés au Pentagone. En revanche, même si le premier a

été fondé, il n'a jamais été opérationnel, et le dernier n'a jamais été fondé. En fin de compte, un sous-comité du Short Range Committee a été formé avec six membres :

- William Selden et Gertrude Tierney de IBM ;
- Howard Bromberg et Howard Discount de RCA ;
- Vernon Reeves et Jean E. Sammet de Sylvania Electric Products.

Ce sous-comité a terminé les spécifications de COBOL fin 1959. Elles étaient largement inspirées par le langage FLOW-MATIC inventé par Grace Hopper, surnommée « la mère du langage COBOL », et par le langage COMTRAN d'IBM, inventé par Bob Bemer.

Ce langage ayant été conçu aux débuts de l'informatique, sa relative complexité rebute nombre de programmeurs de notre époque, ce qui lui a valu deux interprétations ironiques de son acronyme : Complies Only Because Of Luck (fonctionne uniquement par chance) et Completly Obsolete Business Oriented Language (Langage orienté gestion complètement obsolète)[2].

Histoire des standards COBOL

Ces spécifications furent approuvées par le comité complet, puis par le comité exécutif en janvier 1960 et envoyées au bureau d'impression du gouvernement qui les édita et imprima en les nommant COBOL 60. Le langage fut développé en moins de six mois de travail, et il est encore en utilisation cinquante ans plus tard, après plusieurs révisions standardisées par l'ANSI (American National Standards Institute), dont

- COBOL-68 (1968)
- COBOL-74 (1974)
- COBOL-85 (1985) qui témoigne d'un grand pas vers l'adoption de la programmation structurée par l'industrie informatique
- COBOL 2002 introduit la programmation objet, le support de l'Unicode, du XML, etc.

Traits principaux

La totalité des variables et des structures de données utilisées sont définies au début du programme, avant la division procédurale où il y a les instructions. La manière dont sont définies les variables, c'est-à-dire les espaces de stockage temporaire, est très particulière. C'est une structure arborescente définie par une suite de lignes de code. Chaque ligne commence par un nombre qui définit le niveau d'imbrication du champ ou du groupe de variables.

Par exemple :

```
01 NomPrenom.
   05 Prenom PIC X(20).
   05 Nom PIC X(20).
```

qui défini une structure `NomPrenom` contenant les champs `Prenom` et `Nom` sur 20 caractères.

Comme défini dans la spécification originale, COBOL possédait déjà les nombreuses fonctionnalités qui ont fait son succès : d'excellentes capacités d'auto-documentation, des méthodes pratiques de gestion des fichiers et des types de données variés, dont le format est précisé par la clause`PICTURE`. Comme la plupart des autres langages de l'époque, il ne permet pas de définir de variables locales, de fonctions récursives et d'allouer de la mémoire dynamiquement.

La gestion des décimales en COBOL (nombres en virgule fixe), et la maîtrise des arrondis et des dépassements, permettent d'éviter les nombreux problèmes qui arriveraient en utilisant des nombres à virgule flottantepour les calculs financiers.

Il intégre également un générateur de rapports, défini de la même manière que les autres structures de données. Sont intégrées des fonctions de tri, de fusion et de communication. Un module optionnel permettait également une forme de communication inter-processus par file de messages.

Le parti-pris initial de définir un langage de programmation proche du langage naturel (comme pour FLOW-MATIC) devait faciliter, sinon la programmation, du moins l'audit des programmes COBOL par des gestionnaires non-informaticiens. Ce choix a eu pour conséquence une syntaxe complexe (le langage naturel n'est pas simple), avec de nombreux mots réservés, et de nombreuses options (les opérations de gestion ne sont pas simples non plus) qui valent à COBOL une réputation de verbosité, qui n'est pas forcément fondée sur des faits.

Par exemple en Cobol l'instruction

```
ADD montant TO total-jour total-mois total-annee .
```

s'exprimerait, en C ou autres langages dérivés, par

```
total_jour += montant;
total_mois += montant;
total_annee += montant;
```

Comme d'autres langages de l'époque (par exemple Fortran 2), COBOL offrait la possibilité de modifier du code pendant l'exécution à l'aide de la fameuse instruction`ALTER X TO PROCEED TO Y` (altérer X pour aller vers Y). Cette possibilité dangereuse, qui transposait une technique courante de la programmation en langage machine, a été supprimée par la suite.

Les versions successives du standard ont modernisé le langage, par exemple en ajoutant des structures de contrôle améliorées et le support de la programmation objet, tout en préservant au maximum la compatibilité avec les versions précédentes, de façon à éviter d'avoir à modifier l'énorme stock de programmes COBOL en service.

Le poids de l'héritage

Le langage COBOL était de loin le langage le plus employé des années 1960 à 80, et reste toujours en utilisation dans des grandes entreprises (en 2009), notamment dans les institutions financières qui

disposent d'une vaste bibliothèque d'applications COBOL. Écrites à une époque où les octets coûtaient cher, et où l'an 2000 était encore fort loin, ces applications ont fait craindre le fameux bogue de l'an 2000. On redoutait en effet que par mesure d'économie les programmeurs n'aient codé les années sur 2 chiffres plutôt que 4. En réalité, les banques et autres institutions financières géraient depuis très longtemps des dossiers sur 10, 20 voire 30 ans (prêts par exemples) et n'ont pas attendu 1999 pour s'occuper du problème.

En 2005, le Gartner Group estimait que 75% des données du monde des affaires étaient traitées par des programmes en COBOL et que 15% des nouveaux programmes développés le seront dans ce langage.

COBOL permet d'effectuer des traitements comptables du fait de ses capacités arithmétiques en virgule fixe, notamment pour les traitements par lot où il présente d'excellentes performances, à condition que les calculs soient très basiques. Mais, même si les évolutions de COBOL l'ont aujourd'hui doté de certains des outils fournis par les langages modernes (récursivité, allocation dynamique, objets, etc.), son usage reste confiné aux applications de gestion.

Structure d'un programme en COBOL

Un programme comporte quatre divisions. La norme COBOL-85 ne rend obligatoire que la première.

- `IDENTIFICATION DIVISION.`
- : Contient des informations générales sur le programme (dont le nom).
- `ENVIRONMENT DIVISION.`
- : Contient des informations sur l'environnement (matériel et logiciel) dans lequel le programme s'exécute.
- `DATA DIVISION.`
- : Contient les descriptions de données (variables, fichiers, paramètres et parfois description d'écran).
- `PROCEDURE DIVISION.`
- : Contient la description des traitements effectués.

Chaque division est composée de 'sections', formées de 'paragraphes' composés de 'phrases' qui peuvent être des phrases impératives ou des clauses. Chaque phrase doit être terminée par un point.

Les six premières colonnes de chaque ligne de programme sont considérées comme une zone de commentaire, servant autrefois à numéroter les cartes perforées (en cas de chute du paquet, il suffisait de les passer sur une trieuse pour reconstituer la version correcte du programme). La septième colonne contient un caractère de contrôle : espace pour les lignes actives, étoile pour les commentaire.

La huitième colonne est le début des titres de paragraphes.

La douzième colonne est le début des instructions.

Les compilateurs COBOL modernes permettent l'emploi d'un format libre qui n'impose plus le colonnage.

Exemple de programme (Bonjour !)

Écrit dans le style typique des programmes sur cartes perforées (années 1960-70), avec lignes numérotées

```
000100 IDENTIFICATION DIVISION.
000200  PROGRAM-ID.     SALUTTOUS.
000300 DATE-WRITTEN.  21/05/05 19:04.
000400 AUTHOR.
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 DEBUT.
100300     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100400     DISPLAY "BONJOUR !" LINE 15 POSITION 10.
100500     STOP RUN.
```

Note : ERASE EOS signifie "Erase End Of Screen". La commande ligne 100300 a donc pour effet d'effacer l'écran.

Exemple en format libre

Autre version du même exemple en COBOL-85 format libre :

```
Identification division.
 Program-id. Hello.
Procedure division.
 Display "Hello world!" line 15 position 10.
 Stop run.
```

Fortran

Fortran (FORmula TRANslator) est un langage de programmation utilisé principalement en mathématiques et dans les applications de calcul scientifique.

Historique

John Backus, pionnier de l'informatique, publie en 1954 un article titré Preliminary Report, Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN. Il fallut ensuite deux ans d'effort à l'équipe qu'il dirige au sein d'IBM pour écrire le premier compilateur FORTRAN (25 000 lignes, pour l'IBM 704).

Aujourd'hui encore (2007) le langage FORTRAN reste très utilisé, d'une part en raison de la présence de très nombreuses bibliothèques de fonctions utilisables en FORTRAN, d'autre part parce qu'il existe descompilateurs FORTRAN performants qui produisent des exécutables très rapides. Toutefois, beaucoup d'algorithmes même scientifiques sont publiés aujourd'hui (2009) en C et C++[réf. nécessaire], dont les compilateurs sont disponibles sur la plupart des machines..

Le Fortran ayant été créé à l'époque des cartes perforées (en particulier avec le système FMS), optimisait la mise en page de ses sources dans cette optique, jusqu'au Fortran 90. Le code a dû longtemps par exemple commencer à partir de la 7e colonne et ne pas dépasser la 72e.

- La colonne 1 était réservée à l'indicateur "Commentaire" (une lettre C)
- Les colonnes 2 à 5 à une étiquette numérique facultative de l'instruction
- La colonne 6 à l'indicateur "Suite de l'instruction précédente" (souvent un numéro de 1 à 9)
- Les colonnes 73 à 80 à l'identification et la numérotation des cartes perforées (souvent les 3 initiales du projet, du chef de projet ou du programmeur, suivi numéros de 5 chiffres attribués de 10 en 10 pour permettre des insertions de dernière minute).

De nombreux codes industriels ont été écrits depuis longtemps en Fortran et la compatibilité des nouvelles versions avec les précédentes est indispensable, au prix de conserver des notions qui ne s'imposent plus.

Le langage BASIC, dans sa version originale (1964) a été conçu comme un petit langage à caractère pédagogique permettant d'initier les étudiants à la programmation, avant de passer aux langages "sérieux" de l'époque : FORTRAN et Algol. On y retrouve donc quelques traits du langage FORTRAN.

Il existe des extensions libres, basées sur gcc pour compiler les Fortran 77 et maintenant 90 et 95, entre autres sous Linux. Intel fournit aussi un compilateur propriétaire gratuit pour le Fortran 90, pour l'architecture x86 mais uniquement sous Linux. Il est cependant possible d'obtenir une version d'évaluation pour Mac OS X et Windows.

Exemple

```
      PROGRAM DEGRAD
!
! Imprime une table de conversion degrés -> radians
! =================================================
!
! Déclaration des variables
```

```
      INTEGER DEG
      REAL RAD, COEFF
!
! En-tête de programme
      WRITE ( *, 10)
   10 FORMAT       (' ',20('*') /                                    &
    &               ' * Degres * Radians *' /                        &
    &               ' ', 20('*') )
!
! Corps de programme
      COEFF = (2.0 * 3.1416) / 360.0
      DO DEG = 0, 90
         RAD = DEG * COEFF
         WRITE ( *, 20) DEG, RAD
   20 FORMAT          (' *  ',I4,'  * ',F7.5,' *')
      END DO
!
! Fin du tableau
      WRITE ( *, 30)
   30 FORMAT       (' ',20('*') )
!
! Fin de programme
      STOP
      END PROGRAM DEGRAD
```

Notes:

- Ce programme est écrit en Fortran 90.
- Le symbole ! comme premier caractère indique un commentaire.
- La déclaration des variables est facultative en Fortran, mais sans déclaration, la variable DEG serait alors de type REAL (les variables dont le nom commence par une des lettres I, J, K, L, M ou N sont par défaut de type INTEGER, les autres de type REAL).
- L'instruction WRITE se réfère à une unité d'entrée-sortie (ici * désigne le terminal) et une spécification de format. Par exemple le format d'étiquette 20 indique qu'il faut écrire un espace, une étoile et deux espaces, un entier (la valeur de DEG) sur 4 caractères puis la valeur de RAD sur 7 caractères dont 5 après le point décimal et enfin un espace et une étoile. Une déclaration de FORMAT peut être n'importe où ; une habitude est de la mettre juste après le WRITE à laquelle elle se réfère, une autre est de les mettre toutes à la fin de l'unité de programme. Plus d'une instruction WRITE peut faire référence à un même FORMAT.
- Le caractère / à la fin d'une ligne indique une suite à la ligne suivante et le caractère & au début de la ligne indique la suite de la ligne précédente.
- L'instruction "DO DEG = 0,90" indique de répéter en boucle les instructions qui suivent (jusqu'au END DO) pour toutes les valeurs de DEG de 0 à 90 par pas de 1.

Différentes versions de Fortran

- **1956**. FORTRAN II n'avait qu'une seule instruction de branchement ("IF-arithmétique") à 3 adresses : `IF (A-B) 10, 20, 30` indiquait de sauter aux instructions d'étiquette 10, 20 ou 30 selon que A-B était négatif, nul ou positif.
- **1958**. FORTRAN III n'est jamais "sorti" sous forme de produit.
- **1962**. FORTRAN IV a introduit, entre autres, l'instruction "IF-logique", permettant d'écrire `IF (A .GE. B) GOTO 10` (aller à 10 si A est supérieur ou égal à B).
- FORTRAN V était le nom envisagé au départ pour PL/I, langage de programmation universel d'IBM qui devait réunir les meilleurs aspects de Fortran (pour les applications scientifiques), de COBOL (pour les applications de gestion), avec quelques emprunts à Algol.
- **1966**. FORTRAN 66 est la première version officiellement standardisée (par l'American Standards Association) de FORTRAN. On la confond souvent avec FORTRAN IV.
- **1977**. FORTRAN 77, entre autres améliorations, facilite la programmation structurée avec des blocs "IF (...) THEN / ELSE / ENDIF". En 78, une extension introduit DO WHILE / END DO.
- **1990**. FORTRAN 90 : modules, récursivité, surcharge des opérateurs, nouveaux types de données, etc. C'est une mise à jour importante pour mettre FORTRAN au niveau des autres langages modernes. Les restrictions concernant la mise en forme des programmes (colonnes 1 à 7, 72 à 80 ...) disparaissent : l'écriture se fait enfin en format libre.
- **1995**. FORTRAN 95
- **2003**. FORTRAN 2003 : comme son vieux collègue COBOL, Fortran supporte maintenant la programmation orientée objet.
- **2008**. FORTRAN 2008

Assembleur

Un **langage d'assemblage** ou **langage assembleur** ou simplement **assembleur** par abus de langage, abrégé **ASM** est, en programmation informatique, un langage de bas niveau qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des symboles dits « mnémoniques » (du grec mnêmonikos, relatif à la mémoire), c'est-à-dire faciles à retenir. Le programme assembleur convertit ces mnémoniques en langage machine en vue de créer par exemple un fichier exécutable.

Sur les premiers ordinateurs, la tâche d'assemblage était accomplie manuellement par le programmeur.

Particularités de l'assembleur

Un langage spécifique à chaque processeur

Le langage machine est le seul langage qu'un processeur puisse exécuter. Or chaque famille de processeur utilise un jeu d'instructions différent.

Par exemple, un processeur de la famille x86 reconnaît une instruction du type

```
10110000 01100001
```

En langage assembleur, cette instruction est représentée par un équivalent plus facile à comprendre pour le programmeur :

```
movb $0x61,%al
```

Ce qui signifie : « mettre la valeur hexadécimale 61 dans le registre "AL" ».

Ainsi le langage assembleur, représentation exacte du langage machine, est spécifique à chaque architecture de processeur. De plus, plusieurs groupes de mnémoniques ou de syntaxes de langage assembleur peuvent exister pour un seul ensemble d'instructions, créant ainsi des macro-instructions.

Réversibilité du langage machine

Contrairement à un langage de haut niveau, il y a une correspondance un à un (une bijection) entre le code assembleur et le langage machine. Ainsi il est théoriquement possible de traduire le code dans les deux sens sans perte d'information. La transformation du code assembleur en langage machine est accomplie par un programme nommé assembleur, dans l'autre sens par un programme désassembleur. Les opérations s'appellent respectivement assemblage et désassemblage.

En pratique, le désassemblage est un peu plus complexe que cela car lors de la création du code en assembleur on peut affecter des noms aux positions en mémoire, commenter son code, utiliser des macro instructions ou générer du code conditionnel au moment de l'assemblage. Tous ces éléments n'apparaissent pas clairement lors du désassemblage.

Instructions machine

Des opérations de base sont disponibles dans la plupart des jeux d'instructions

- déplacement
  - chargement d'une valeur dans un registre
  - déplacement d'une valeur depuis un emplacement mémoire dans un registre, et inversement
- calcul

    - addition, ou soustraction des valeurs de deux registres et chargement du résultat dans un registre
    - combinaison de valeurs de deux registres suivant une opération booléenne (ou opération bit à bit)
- modification du déroulement du programme
    - saut à un autre emplacement dans le programme (normalement, les instructions sont exécutées séquentiellement, les unes après les autres)
    - saut à un autre emplacement, mais après avoir sauvegardé l'instruction suivante afin de pouvoir y revenir (point de retour)
    - retour au dernier point de retour
- comparaison
    - comparer les valeurs de deux registres

Et on trouve des instructions spécifiques avec une ou quelques instructions pour des opérations qui auraient dû en prendre beaucoup. Exemples :

- déplacement de grands blocs de mémoire
- multiplication, division
- arithmétique lourde (sinus, cosinus, racine carrée, opérations sur des vecteurs)
- application d'une opération simple (par exemple, une addition) à un ensemble de données par l'intermédiaire des extensions MMX ou SSE des nouveaux processeurs.

Directives du langage assembleur

En plus de coder les instructions machine, les langages assembleur ont des directives supplémentaires pour assembler des blocs de données et assigner des adresses aux instructions en définissant des étiquettes ou labels.

Ils sont capables de définir des expressions symboliques qui sont évaluées à chaque assemblage, rendant le code encore plus facile à lire et à comprendre.

Ils ont habituellement un langage macro intégré pour faciliter la génération de codes ou de blocs de données complexes.

Exemples simples

Voici quelques exemples simples :

- en syntaxe AT&T (écrits pour l'assembleur GNU (GAS) pour Linux)
- utilisant le jeu d'instructions i386
- à utiliser comme suit:

```
$ gcc truc.S -c -o truc.o
$ ld truc.o -o truc
```

```
$ ./truc
```

Afficher Bonjour

(les commentaires se trouvent après les points-virgule)

```
        .global _start
 BONJ:  .ascii  "Bonjour\n"      ; Définition en mémoire de la chaîne à
afficher. \n correspond au saut de ligne
 _start: mov    $4      , %eax   ; Mettre 4 dans le registre eax (appel
système '''Write'')
        mov     $1      , %ebx   ; Mettre 1 dans le registre ebx
(descripteur de fichier ''STDOUT'')
        mov     $BONJ   , %ecx   ; Mettre l'adresse mémoire de notre
chaîne de caractère dans le registre ecx
        mov     $8      , %edx   ; Mettre la taille de la chaîne dans edx
        int     $0x80            ; Interruption 0x80, exécutant un appel
système sous Linux)

        mov     $1      , %eax   ; Mettre 1 dans eax (appel système
''Exit'')
        mov     $0      , %ebx   ; Mettre 0 dans ebx (valeur de retour du
programme)
        int     $0x80            ; Interruption 0x80, exécutant un appel
système sous Linux)
```

Lire le clavier (16 caractères max) puis l'afficher

```
 # define N 16

        .global _start

        .comm   BUFF    , N

 _start: mov    $3      , %eax
        mov     $0      , %ebx
        mov     $BUFF   , %ecx
        mov     $N      , %edx
        int     $0x80

        mov     %eax    , %edx
        mov     $4      , %eax
        mov     $1      , %ebx
        mov     $BUFF   , %ecx
        int     $0x80

        mov     $1      , %eax
        mov     $0      , %ebx
        int     $0x80
```

Usage du langage assembleur

Il y a des débats sur l'utilité du langage assembleur. Dans beaucoup de cas, des compilateurs-optimiseurs peuvent transformer du langage de haut niveau dans un code qui tourne de façon presque aussi efficace qu'un code assembleur écrit à la main, tout en restant beaucoup plus facile (et moins coûteux) à écrire, à lire et à maintenir.

Cependant,

1. quelques calculs complexes écrits directement en assembleur, en particulier sur des machines massivement parallèles, seront plus rapides, les compilateurs n'étant pas encore assez évolués pour tirer partie des spécificités de ces architectures.
2. certaines routines (drivers) sont parfois plus simples à écrire en langage de bas niveau.
3. des tâches très dépendantes du système, exécutées dans l'espace mémoire du système d'exploitation sont parfois difficiles à écrire dans un langage de haut niveau.

Certains compilateurs transforment, lorsque leur option d'optimisation la plus haute n'est pas activée, des programmes écrits en langage de haut niveau en code assembleur, chaque instruction de haut niveau se traduisant en une série d'instructions assembleur rigoureusement équivalentes et utilisant les mêmes symboles ; cela permet de voir le code dans une optique de débogage et de profilage, ce qui permet de gagner parfois beaucoup plus de temps en remaniant un algorithme. En aucun cas ces techniques ne peuvent être conservées pour l'optimisation finale.

La programmation des systèmes embarqués, souvent à base de microcontrôleurs, est une "niche" traditionnelle pour la programmation en assembleur. En effet ces systèmes sont souvent très limités en ressources (par exemple un microcontrôleur PIC 16F84 est limité à 1024 instructions de 14 bits, et sa mémoire vive contient 136 octets). et requièrent donc une programmation de bas-niveau très optimisée pour en exploiter les possibilités. Toutefois, l'évolution du matériel fait que les composants de ces systèmes deviennent de plus en plus puissants à un coût et à une consommation électrique constants, l'investissement dans une programmation "tout assembleur" beaucoup plus coûteuse en heures de travail devient alors un non-sens en termes d'efforts.

Macro-assembleur

Beaucoup d'assembleurs gèrent un langage de macros. Il s'agit de regrouper plusieurs instructions afin d'avoir un enchaînement plus logique et moins fastidieux.
Par exemple (en assembleur Microsoft MASM) :

```
putchar Macro   car          ; Prototype de la macro
        ifdef   car          ; si car est défini
        mov     dl,car       ;   le mettre dans dl
        endif
        mov     ah,2         ; ah=2 : fonction "putchar" en DOS
        int     21h          ; appel au DOS
```

```
        endm                    ; fin macro
```

est une macro qui affiche un caractère sous MS-DOS. On l'utilisera par exemple ainsi :

```
        putchar "X"
```

Et cela générera :

```
        mov     dl,"X"
        mov     ah,2
        int     21h
```

L4G

Un **L4G** ou **langage de quatrième génération** est un langage de programmation ayant un haut niveau d'abstraction. Ils sont généralement utilisés pour les applications de gestion. Un L4G doit offrir :

- un langage déclaratif de manipulation de données ;
- un langage impératif simple (procédural, fonctionnel ou orienté objet) ;
- un langage de description d'interfaces graphiques avec idéalement un éditeur d'interface ;
- un langage de description de rapports imprimables avec idéalement un éditeur de rapport ;
- un langage de programmation événementielle avec idéalement une liaison entre le code et l'interface graphique.

Les différentes générations

Les langages de première génération s'adressaient aux ordinateurs en langage binaire (des 0 et des 1).

La seconde génération, le langage assembleur, s'adresse au microprocesseur instruction par instruction.

La troisième génération introduit une syntaxe et des mots réservés, ce sont les langages procéduraux (COBOL, Fortran, BASIC, Pascal, langage C, RPG) ou encore à objets (Java, C++, Eiffel).

La quatrième génération, souvent associée à des bases de données, se situe un niveau au-dessus, en intégrant la gestion de l'interface utilisateur et en proposant un langage moins technique, plus proche de la syntaxe naturelle.

Exemples

- Microsoft Access
- Visual FoxPro
- Clarion
- OMNIS Studio

- PowerBuilder
- ADELIA
- Magic eDeveloper
- 4D
- Linotte
- WLangage
- Statistical Analysis System (SAS)

Microsoft Access

Microsoft Access ou MS Access (officiellement Microsoft Office Access) est un L4G comprenant un système de gestion de base de données relationnellesédité par Microsoft. Il fait partie de la suite bureautique MS Office Pro.

MS Access est composé de deux programmes : le moteur de base de données et l'éditeur graphique. Les deux ne sont pas librement séparables.

Principales caractéristiques du produit

MS Access est un logiciel utilisant des fichiers au format Access (extension de fichier mdb pour Microsoft DataBase (extension *.accdb depuis la version 2007)). Il est compatible avec les requêtes SQL (sous certaines restrictions) et dispose d'une interface graphique pour saisir les requêtes (QBE - Query par exemple). Il permet aussi de configurer, avec des assistants ou librement, des formulaires et sous-formulaires de saisie, des états imprimables (avec regroupements de données selon divers critères et des totalisations, sous-totalisations, conditionnelles ou non), des pages html liées aux données d'une base, des macros et des modules VBA.

Comme beaucoup de systèmes de gestion de base de données relationnelles, ses données peuvent être utilisées dans des programmes écrits dans divers langages.

Les langages couramment utilisés avec Access sont le Visual Basic for Application (VBA) et les langages qui disposent de modules d'accès aux données pour les fichiers .mdb : Delphi de Borland, Visual Basic,C++ sous Visual Studio de Microsoft par exemple. VBA, intégré à Access comme à toutes les applications de la suite Microsoft Office, permet de créer des applications de gestion complètes, livrées avec un programme d'installation qui gère automatiquement la mise en place éventuelle d'un runtime d'Access, et dont le code source est protégé dans une version semi-exécutable des fichiers (mde). Il est en effet possible (et conseillé) d'installer un fichier mdb, contenant les tables de données, sur un serveur (ou un poste de réseau poste à poste dédié comme tel) et des

fichiers mdb (ou mde) contenant tous les éléments de l'application sur les postes clients. Dans ce cas, les fichiers clients sont « attachés » aux tables du fichier « mdb » installé sur le poste serveur.

D'après Microsoft, MS Access supporterait des configurations de 256 postes. Mais en pratique, pour une utilisation confortable, MS Access serait limité à une vingtaine d'utilisateurs simultanés, les échanges réseaux étant 10 à 20 fois plus gourmands en ressources qu'avec Microsoft SQL Server par exemple. Le confort d'utilisation de MS Access en réseau peut-être considérablement accru quand on utilise judicieusement les requètes de type Snapshot (lecture seule, modifications ultérieures non visibles) et les requètes de type Dynaset (lecture-écriture) qui sont beaucoup plus gourmandes en ressources réseau. Ces perfectionnements ne permettent pas de dépasser pratiquement une quarantaine d'utilisateurs en simultané. Il est important de noter que ces caractéristiques conviennent largement à la plupart des petites et moyennes entreprises.

D'un point de vue concret Access (avec ses versions 2000 à 2003) convient bien à des applications faisant intervenir jusqu à une centaine de tables (principales et de jointures / relations) avec un maximum pratique de 100 000 enregistrements pour les tables principales et de 1 000 000 d enregistrements pour les tables de jointures (appelées aussi tables de liaisons ou de relations).

Pour une utilisation de plus grande envergure, il peut servir de client pour un serveur de bases de données (comme SQL Server, Oracle, MySQL, etc.) via ODBC ou OLE DB, on parle d'utilisation frontale. Si Access est limité en nombre d'utilisateurs, il peut par contre gérer, sur de petits réseaux locaux de bonne qualité technique, des quantités d'informations qui vont bien au-delà des besoins de beaucoup d'organismes. Exemples : une base de données des pièces automobiles de toutes les marques pour des garages, ou une comptabilité de PME sur un seul site, mais autorisant des consolidations de plusieurs sites autonomes.

Les données d'Access sont facilement exploitables dans les publipostages de Word et les tableaux Excel. Réciproquement les feuilles de données d'Excel peuvent être « attachées », comme une des tables de la base de données ou importées ponctuellement dans une table Access.

Par le biais du language de programmation VBA il est possible, depuis l'intérieur d'Access, de construire ou de modifier tout type de tableaux Excel, puis de traiter et transférer les informations adéquat de la base de données vers ce même tableur. Attention une fois Excel lancé depuis Access, il faut le fermer proprement. Sinon l'application peut continuer à tourner en tâche de fond, sans être visible de l'utilisateur.

En Java, Microsoft Access peut être utilisé de façon transparente via JDBC à l'aide de la passerelle JDBC-ODBC de Sun.

Les différentes versions

Lancement en 1992

- Access 1

- Access 1.1
- Access 2
- Access 95 (7)
- Access 97 (8)
- Access 2000 (9)
- Access XP 2002 (10)
- Access 2003 (11)
- Access 2007 (12)

arrivera bientôt Access unlimited tools (en pleine création)

Algorithmique

On désigne par **algorithmique** l ensemble des activités logiques qui relèvent des **algorithmes** ; en particulier, en informatique, cette discipline désigne l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception des algorithmes. Le mot vient du nom du mathématicien Al Khuwarizmi (latinisé au Moyen Âge en Algoritmi), qui, au IXe siècle écrivit le premier ouvrage systématique sur la solution des équations linéaires et quadratiques. Dans le cas général, l algorithmique s effectue au moyen de calculs.

Il est parfois fait usage du mot **algorithmie**, bien que ce dernier ne figure pas dans la plupart des dictionnaires.

Définition

Un algorithme est un processus systématique de résolution, par le calcul, d'un problème permettant de présenter les étapes vers le résultat à une autre personne physique (un autre humain) ou virtuelle (un calculateur). En d'autres termes, un algorithme est un énoncé dune suite d opérations permettant de donner la réponse à un problème. Si ces opérations sexécutent en séquence, on parle d algorithme séquentiel. Si les opérations sexécutent sur plusieurs processeurs en parallèle, on parle d algorithme parallèle. Si les tâches sexécutent sur un réseau de processeurs on parle dalgorithme réparti ou distribué.

Historique

Antiquité

Les algorithmes dont on a retrouvé des descriptions exhaustives ont été utilisés dès lépoque des Babyloniens, pour des calculs concernant le commerce et les impôts.

L algorithme le plus célèbre est celui qui se trouve dans le livre 7 des Éléments d'Euclide. Il permet de trouver le plus grand diviseur commun, ou PGCD, de deux nombres. Un point particulièrement remarquable est qu il contient explicitement une itération et que les propositions 1 et 2 démontrent (maladroitement pour nos contemporains) sa convergence.

Étude systématique

L algorithmique a été systématisée par le mathématicien perse Al Khuwarizmi (né vers 780 - mort vers 850), auteur d un ouvrage (souvent traduit par Lalgèbre et le balancement) qui décrit des méthodes de calculs algébriques (en plus d'introduire le zéro des Indiens).

Le savant arabe Averroès (1126-1198) évoque une méthode de raisonnement où la thèse saffine étape par étape (itérativement) jusqu à une certaine convergence et ceci conformément au déroulement d un algorithme. À la même époque, au XIIe siècle, le moine Adelard de Bath a introduit le terme latin de algorismus (par référence au nom de Al Khuwarizmi). Ce mot donne algorithme en français en 1554.

Au XVIIe siècle, on pourrait entrevoir une certaine allusion à la méthode algorithmique chez René Descartes dans la méthode générale proposée par le Discours de la méthode (1637), notamment quand, en sa deuxième partie, le logicien français propose de « diviser chacune des difficultés que j examinerois, en autant de parcelles quil se pourroit, et qu il seroit requis pour les mieux résoudre. » Sans évoquer explicitement les concepts de boucle ou ditération, l approche de Descartes prédispose la logique à accueillir le concept de programme, mot qui naît en français en 1677.

L utilisation du terme algorithme a été remarquable chez Ada Lovelace, fille de Lord Byron et assistante de Charles Babbage (1792-1871).

Vocabulaire

Le substantif algorithmique désigne la méthode utilisant des algorithmes. Le terme est également employé comme adjectif.

Un algorithme énonce une résolution sous la forme dune série d opérations à effectuer. La mise en uvre de l algorithme consiste en lécriture de ces opérations dans un langage de programmation et constitue alors la brique de base dun programme informatique.

Les informaticiens utilisent fréquemment langlicisme implémentation pour désigner cette mise en uvre. L écriture en langage informatique est aussi fréquemment désignée par le terme « codage »,

qui n a ici aucun rapport avec la cryptographie, mais qui se réfère au terme « code source » pour désigner le texte, en langage de programmation, constituant le programme. Lalgorithme devra être plus ou moins détaillé selon le niveau dabstraction du langage utilisé ; autrement dit, une recette de cuisine doit être plus ou moins détaillée en fonction de lexpérience du cuisinier.

Exemples d algorithmes

Il existe un certain nombre d algorithmes classiques, utilisés pour résoudre des problèmes ou plus simplement pour illustrer des méthodes de programmation. On se référera aux articles suivants pour de plus amples détails :

- tours de Hanoï, problème célèbre illustrant la programmation récursive ;
- algorithme de tri, ou comment trier un ensemble de nombres le plus rapidement possible ;
- huit dames, placer huit dames sur un échiquier sans quelles puissent se prendre entre elles ;
- algorithme récursif, quelques présentations dalgorithmes récursifs simples ;
- algorithme du simplexe, qui minimise une fonction linéaire de variables réelles soumises à des contraintes linéaires.
- Fraction continue d'un nombre quadratique, permettant d'extraire une racine carrée.

Complexité algorithmique

Article détaillé : Théorie de la complexité des algorithmes.

Les principales notions mathématiques dans le calcul du coût dun algorithme précis sont les notions de domination (notée O(f(n)), « grand o »), où f est une fonction mathématique de n, variable désignant la quantité d informations (en bits, en nombre denregistrements, etc.) manipulée dans l algorithme. En algorithmique on trouve souvent des complexités du type :

Notation	Type de complexité
$O(1)$	complexité constante (indépendante de la taille de la donnée)
$O(log(n))$	complexité logarithmique
$O(n)$	complexité linéaire
$O(nlog(n))$	complexité quasi-linéaire

$O(n^2)$	complexité quadratique
$O(n^3)$	complexité cubique
$O(n^p)$	complexité polynomiale
$O(n^{\log(n)})$	complexité quasi-polynomiale
$O(2^n)$	complexité exponentielle
$O(n!)$	complexité factorielle

Sans entrer dans les détails mathématiques, le calcul de lefficacité dun algorithme (sa complexité algorithmique), consiste en la recherche de deux quantités importantes. La première quantité est l évolution du nombre dinstructions de base en fonction de la quantité de données à traiter (par exemple, pour un algorithme de tri, il s'agit du nombre de données à trier), que lon privilégiera sur le temps d'exécution mesuré en secondes (car ce dernier dépend de la machine sur laquelle l'algorithme s'exécute). La seconde quantité estimée est la quantité de mémoire nécessaire pour effectuer les calculs. Baser le calcul de la complexité dun algorithme sur le temps ou la quantité effective de mémoire qu un ordinateur particulier prend pour effectuer ledit algorithme ne permet pas de prendre en compte la structure interne de lalgorithme, ni la particularité de lordinateur : selon sa charge de travail, la vitesse de son processeur, la vitesse daccès aux données, lexécution de l algorithme (qui peut faire intervenir le hasard) ou son organisation de la mémoire, le temps dexécution et la quantité de mémoire ne seront pas les mêmes.

Il existe également un autre aspect de l'évaluation de l'efficacité d'un algorithme : les performances en moyenne de cet algorithme. Elle suppose d'avoir un modèle de la répartition des données de l'algorithme, tandis que la mise en uvre des techniques d'analyse implique des méthodes assez fines de combinatoire et d'évaluation asymptotique, utilisant en particulier les séries génératrices et des méthodes avancées d'analyse complexe. L'ensemble de ces méthodes sont regroupées sous le nom de combinatoire analytique.

On trouvera dans l article sur la théorie de la complexité des algorithmes dautres évaluations de la complexité qui vont en général au-delà des valeurs proposées ci-dessus et qui répartissent les problèmes (plutôt que les algorithmes) en classes de complexité.

Quelques indications sur l efficacité des algorithmes

Souvent, l efficacité d un algorithme n est connue que de manière asymptotique, cest-à-dire pour de grandes valeurs du paramètre n. Lorsque ce paramètre est suffisamment petit, un algorithme de complexité supérieure peut en pratique être plus efficace. Ainsi, pour trier un tableau de 30 lignes (c est un paramètre de petite taille), il est inutile dutiliser un algorithme évolué comme le Tri rapide (l un des algorithmes de tri les plus efficaces en moyenne) : lalgorithme de tri le plus trivial sera suffisamment efficace.

Entre deux algorithmes dont la complexité est identique, on cherchera à utiliser celui dont loccupation mémoire est la plus faible. L analyse de la complexité algorithmique peut également servir à évaluer l occupation mémoire d un algorithme. Enfin, le choix dun algorithme plutôt qu un autre doit se faire en fonction des données que l on s attend à lui fournir en entrée. Ainsi, le Quicksort (ou tri rapide), lorsque l on choisit le premier élément comme pivot, se comporte de façon désastreuse si on l applique à une liste de valeurs déjà triée. Il nest donc pas judicieux de lutiliser si on prévoit que le programme recevra en entrée des listes déjà presque triées.

Un autre paramètre à prendre en compte est la localité de lalgorithme. Par exemple pour un système à mémoire virtuelle qui dispose de peu de mémoire (par rapport au nombre de données à traiter), le Tri rapide sera normalement plus efficace que le Tri par tas car le premier ne passe quune seule fois sur chaque élément de la mémoire tandis que le second accède à la mémoire de manière discontinue (ce qui augmente le risque de swapping).

Enfin, il existe certains algorithmes dont la complexité est dite amortie. Cela signifie que, pour certaines exécutions de l algorithme (cas marginaux), la complexité de lalgorithme sera très supérieure au cas moyen. Bien sûr, on n utilise la notion de complexité amortie que dans les cas où cette réaction est très marginale.

Les heuristiques

Pour certains problèmes, les algorithmes ont une complexité beaucoup trop grande pour obtenir un résultat en temps raisonnable, même si lon pouvait utiliser une puissance de calcul phénoménale. On est donc amené à rechercher une solution la plus proche possible dune solution optimale en procédant par essais successifs. Puisque toutes les combinaisons ne peuvent être essayées, certains choix stratégiques doivent être faits. Ces choix, généralement très dépendants du problème traité, constituent ce qu on appelle une heuristique. Le but d une heuristique n'est donc pas d'essayer toutes les combinaisons possibles afin de trouver celle répondant au problème, mais de trouver une solution approchée convenable (qui peut être exacte dans certains cas) dans un temps raisonnable. Cest ainsi que les programmes de jeu déchecs, de jeu de go (pour ne citer que ceux-là) font appel de manière très fréquente à des heuristiques qui modélisent lexpérience d un joueur. Certains logiciels antivirus se basent également sur des heuristiques pour reconnaître des virus informatiques non répertoriés dans leur base, en sappuyant sur des ressemblances avec des virus connus.

Applications

- Algorithme génétique en informatique décisionnelle
- Allocation de mémoire et ramasse-miettes
- Cryptologie et compression de données
- Informatique musicale
- Structure de données, Algorithmes de tri et Recherche dichotomique

Entrées-sorties

Dans un système à base d'un processeur, d'un microprocesseur, d'un microcontrôleur ou d'un automate, on appelle **Entrées-Sorties** les échanges d'informations entre le processeur et les périphériques qui lui sont associés. De la sorte, le système peut réagir à des modifications de son environnement, voire le contrôler. Elles sont parfois désignées par l'acronyme **I/O**, issu de l'anglais Input/Output.

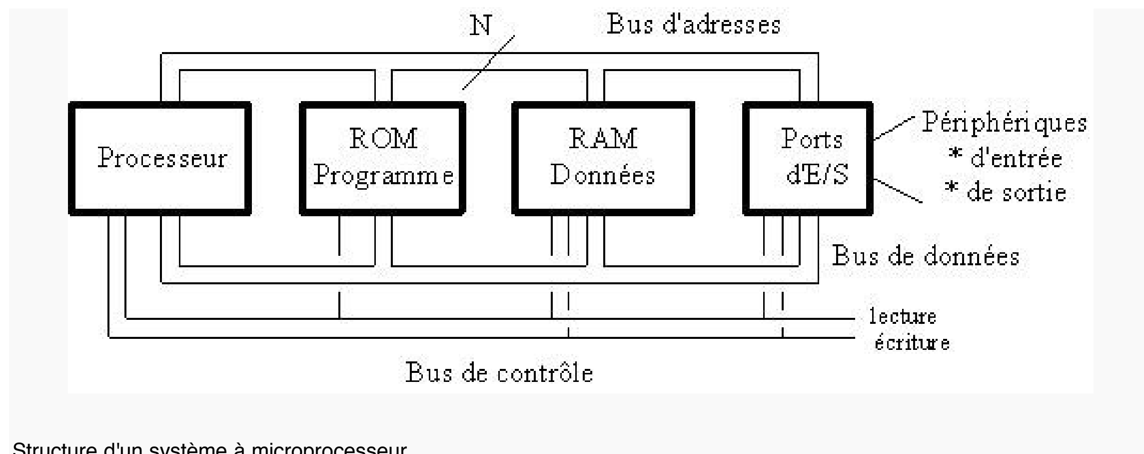
Dans un système d'exploitation,

- Les entrées sont les données envoyées par un périphérique (disque, réseau, clavier) à destination de l'unité centrale ;
- Les sorties sont les données émises par l'unité centrale à destination d'un périphérique (disque, réseau, écran...).

Exemple simplifié :

- taper sur les touches du clavier envoie une série de codes vers le processeur ; ces codes sont considérés comme des données d'entrée ;
- le processeur affiche les résultats du traitement des données sur un écran ; ce sont des données de sortie. Habituellement, l'écran est géré par un programme de gestion d'affichage.

Structure d'un système à microprocesseur



Structure d'un système à microprocesseur

Un système à microprocesseur comporte nécessairement les éléments suivants :

- un **processeur**, qui est le cerveau du système ; il est capable d'effectuer des opérations arithmétiques et logiques et d'organiser des transferts de données entre les différents éléments du système ;
- une zone de **mémoire morte** (ROM, EPROM, EPROM Flash) qui stocke le programme ;
- une zone de **mémoire vive** (RAM) qui stocke les données pendant l'exécution du programme ; le contenu de cette mémoire est perdu lorsqu'on coupe l'alimentation du système ;
- des **périphériques** ; leur nombre et genre dépendent de l'application.

Les différents éléments du système sont reliés par 3 bus :

- le bus de données permet, comme son nom l'indique, la circulation des données, mais aussi des instructions, entre les 4 grands blocs ;
- le bus d'adresse permet au processeur de désigner à chaque instant la case mémoire ou le périphérique auquel il veut faire appel ;
- le bus de contrôle est également géré par le processeur et indique, par exemple, s'il veut faire une écriture ou une lecture dans une case mémoire, ou une entrée/sortie de ou vers un périphérique ; on trouve également, dans le bus de contrôle, une ou plusieurs lignes qui permettent aux circuits périphériques d'effectuer des demandes au processeur ; ces lignes sont appelées lignes d'interruptions matérielles (IRQ).

L'évolution de la technologie fait que des systèmes qui, précédemment, nécessitaient plusieurs boîtiers, peuvent parfaitement être intégrés dans un seul boîtier qui regroupe les différentes fonctions ; voir par exemple la famille de processeurs ADuC d'Analog Devices.

Exemple de système à microprocesseur

Une machine à laver est un excellent exemple de système de contrôle piloté par microprocesseur. Les principaux éléments de la machine sont :

- un tambour dans lequel sera placé le linge à laver ;
- un moteur pour faire tourner ce tambour à vitesse plus ou moins grande selon la phase du programme (lavage, essorage) ;
- une résistance chauffante pour chauffer l'eau ;
- une électro-vanne pour autoriser l'entrée de l'eau de la distribution dans la cuve de lavage au début du cycle ;
- une pompe pour vider l'eau en fin de cycle ;
- un détecteur de niveau d'eau pour arrêter le remplissage de la cuve ;
- un thermomètre électronique pour arrêter le chauffage lorsque l'eau a atteint la température désirée ;
- un ou plusieurs commutateurs pour sélectionner le programme, la température de l'eau, la vitesse d'essorage ;
- un bouton de mise en marche et d'arrêt de la machine ;
- un ou plusieurs voyants ou indicateurs (témoin de marche, état d'avancement du programme).

Le processeur va recevoir des informations des périphériques d'entrée :

- commutateur(s) ;
- détecteur de niveau ;
- thermomètre électronique.

En fonction de ces informations, il va envoyer des commandes aux périphériques de sortie :

- moteur ;
- résistance chauffante ;
- électro-vanne ;
- pompe ;
- voyants et indicateurs.

Ports d'entrées/sorties

Les périphériques sont reliés au reste du système par des circuits appelés ports d'entrées et ports de sortie (certains ports peuvent combiner les deux fonctions).

Un port d'entrée est essentiellement composé de tampons trois états. Ceux-ci se comportent comme des interrupteurs électroniques qui font apparaître, au moment voulu, les niveaux logiques du

périphérique d'entrée (choisi par le bus d'adresse) sur le bus de données ; ces niveaux seront mémorisés dans un registre du processeur (le registre est une case de RAM).

Un port de sortie est essentiellement composé de bascules de type D. Celles-ci se comportent comme des petites mémoires. Leur entrée est reliée au bus de données. Le processeur vient écrire un niveau logique 0 ou 1 dans chacun des bascules. Les sorties des bascules contrôlent les périphériques, généralement via un étage de puissance.

Périphériques d'entrée

Une entrée est un flux de données provenant soit :

- du réseau,
- d'une lecture d'information sur disque,
- d'une saisie clavier, d'un mouvement de souris, d'un crayon optique
- ou de tout autre périphérique prévu pour interagir avec un système informatique.

Ces signaux d'entrée génèrent des Interruptions matérielles qui sont traitées en priorité par le gestionnaire d'interruptions du noyau du système d'exploitation.

Dans les systèmes à microprocesseurs, tels la machine à laver évoquée ci-dessus, on trouve des boutons poussoirs, des commutateurs.

De nombreux microcontrôleurs incorporent des compteurs ; les signaux mis en forme et appliqués aux entrées de comptage constituent aussi des signaux d'entrée du système.

Dans les systèmes informatiques, le choix est bien plus vaste : clavier, souris, crayon optique, numériseur, convertisseurs analogiques/numériques

Insistons sur le fait que, pour être traités par le processeur, les signaux, quels qu'ils soient, doivent être convertis en signaux logiques compatibles avec le processeur. Dans certains cas, il faudra donc placer des convertisseurs de niveau ou des étages d'isolement (souvent des opto-coupleurs).

Périphériques de sortie

Les sorties sont associées à des trappes ou appels systèmes.

Une sortie peut être (cette liste n'est pas exhaustive) :

- un signal (électrique, onde) ;
- un flux de données (réseau), une écriture sur disque ou une mise en mémoire ;
- un affichage, un son.

Dans les systèmes à microprocesseurs, on utilise des diodes électroluminescentes (DELs) ou des ampoules à incandescence comme voyants ou indicateurs, des afficheurs numériques ou alphanumériques à DELs ou à cristaux liquides pour l'affichage des messages du système, des relais

(pour commander des charges nécessitant des courants et/ou des tensions élevés), des optocoupleurs

Dans les systèmes informatiques, le choix est vaste : écran pour l'affichage, imprimante pour la production de documents sur papier, convertisseurs numériques/analogiques

Périphériques d'entrées/sorties

Un grand nombre de périphériques sont à la fois des périphériques d'entrée et de sortie. Le modem, par exemple, permet d'envoyer ou de recevoir des informations en provenance du monde extérieur : courrier électronique, navigation Internet, mais aussi envoi et réception de fax, téléphonie par ordinateur (VoIP, Voice over IP).

Les cartes réseau permettent de relier entre eux plusieurs ordinateurs afin de réaliser un réseau local d'ordinateurs, ce qui permet de partager des fichiers ou des ressources telles une imprimante réseau, un numériseur

Et puis, il y a toute la gamme des mémoires de masse : disque dur, lecteur de disquette, carte mémoire.

Gestion des entrées/sorties

On distingue principalement trois façons de gérer les entrées/sorties.

Entrées/sorties programmées

Pendant l'exécution de son programme principal, le microprocesseur va périodiquement lire l'état des périphériques d'entrée et modifie, si nécessaire, l'état des ports de sortie. C'est la technique la plus simple. Exemple : système de régulation de chauffage d'un bâtiment.

Interruptions

Cette technique est utilisée lorsque le processeur doit réagir rapidement à un changement d'état d'un port d'entrée. Le périphérique prévient le processeur par une ligne d'interruption prévue à cet effet. Le processeur interrompt la tâche en cours, saute dans le sous-programme destiné à gérer la demande spécifique qui lui est adressée ; à la fin du sous-programme, le processeur reprend l'exécution du programme principal là où il l'avait laissée.

Accès direct à la mémoire

Cette technique, connue souvent par ses initiales DMA (Direct Memory Access), est utilisée lorsque l'on doit procéder à un transfert rapide d'un grand nombre de données entre, par exemple, un lecteur de CD et un disque dur. Plutôt que de transférer les octets d'abord vers un registre du processeur, puis seulement vers le disque dur, les octets sont transférés directement d'un périphérique à l'autre sans passer par les registres du processeur. Le transfert des données est organisé par un circuit spécial appelé contrôleur d'interruptions, qui prend la place du processeur pendant le transfert et gère les bus d'adresses et de contrôle.

Performances

Les performances d'un ordinateur mesurent le temps qui lui est nécessaire pour effectuer un traitement donné. Trois éléments influencent ces performances :

- la puissance du processeur ;
- la mémoire disponible ;
- le temps consacré aux opérations d'entrées/sorties.

Le temps nécessaire pour un traitement informatique quel qu'il soit est toujours déterminé par un de ces trois éléments mais celui des entrées/sorties est généralement prépondérant. En effet, le temps consacré aux opérations I/O se compte en millisecondes alors que celui consacré aux instructions effectuées par le processeur se compte en nanosecondes.

La taille de la mémoire est surtout importante dans la mesure où elle permet de réduire le nombre d'opérations d'entrées/sorties, soit parce qu'une part plus importante des programmes applicatifs peut résider en mémoire, réduisant ainsi les phénomènes de pagination, soit parce que une partie de cette mémoire peut-être utilisée comme tampon (mémoire cache) pour le stockage des flux de données des opérations I/O.

En programmation comme au niveau système (par exemple sur les mainframes), deux éléments matériels (entre autres) influencent les performances des entrées/sorties, c'est-à-dire leur vitesse :

- la charge du processeur (i.e. son taux d'occupation), qui fournit les données sortantes ou traite les données entrantes ;
- la charge du dispositif d'entrée/sortie, qui émet ou reçoit les données (on parle généralement des lectures/écritures notamment pour les accès disques).

Si les ressources CPU ou I/O sont insuffisantes lors de l'exécution d'un ou plusieurs traitements simultanés, on parle de saturation.

Compilateur

Un **compilateur** est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible, en préservant la signification du texte source. Ce schéma général décrit un grand nombre de programmes différents ; et ce que l'on entend par « signification du texte source » dépend du rôle du compilateur. Lorsque l'on parle de compilateur, on suppose aussi en général que le langage source est, pour l'application envisagée, de plus haut niveau que le langage cible, c'est-à-dire qu'il présente un niveau d'abstraction supérieur.

En pratique, un compilateur sert le plus souvent à traduire un code source écrit dans un langage de programmation en un autre langage, habituellement un langage d'assemblage ou un langage machine. Le programme en langage machine produit par un compilateur est appelé code objet.

Le premier compilateur, A-0 System, a été écrit en 1951 par Grace Hopper.

Structure d'un compilateur

La tâche principale d'un compilateur est de produire du code objet correct. La plupart des compilateurs permettent d'optimiser le code (le code objet optimisé s'exécutera plus rapidement, ou aura une occupation mémoire moindre).

Un compilateur fonctionne par analyse-synthèse, c'est-à-dire qu'au lieu de remplacer chaque construction du langage source par une suite équivalente de constructions du langage cible, il commence par analyser le texte source pour en construire une représentation intermédiaire qu'il traduit à son tour en langage cible.

Il est donc naturel de séparer au moins conceptuellement, mais aussi en pratique le compilateur en une partie avant (ou frontale), parfois appelée « souche », qui lit le texte source et produit la représentation intermédiaire, et une partie arrière (ou finale), qui parcourt cette représentation pour produire le texte cible. Dans un compilateur idéal, la partie avant est indépendante du langage cible, tandis que la partie arrière est indépendante du langage source. Certains compilateurs effectuent de plus sur la forme intermédiaire des traitements substantiels, que l'on peut regrouper en une partie centrale, indépendante à la fois du langage source et de la machine cible. On peut ainsi écrire des compilateurs pour toute une gamme de langages et d'architectures en partageant la partie centrale, à laquelle on attache une partie avant par langage et une partie arrière par architecture.

Les étapes de la compilation incluent

- le découpage du programme en lexèmes (analyse lexicale) ;
- la vérification de la correction de la syntaxe du programme (analyse syntaxique) ;
- l'analyse des structures de données (analyse sémantique) ;
- la transformation du code source en code intermédiaire ;
- l'application de techniques d'optimisation sur le code intermédiaire ;
- l'allocation de registres et la traduction du code intermédiaire en code objet, avec éventuellement l'insertion de données de débogage et d'analyse de l'exécution ;
- et enfin l'édition des liens.

Ces différentes étapes expliquent que les compilateurs fassent toujours l'objet de recherches, particulièrement dans le domaine de l'optimisation du code produit.

La plupart (mais pas tous) des compilateurs traduisent un fichier source d'un programme écrit dans un langage de programmation en un fichier objet (ou un exécutable, ou un fichier en assembleur, ou même en un autre langage).

Une implémentation (réalisation concrète) d'un langage de programmation peut être interprétée ou compilée. C'est cette réalisation qui est un compilateur ou un interpréteur, et un langage de programmation (spécification plus ou moins théorique et formalisée de sa syntaxe et de sa sémantique) peut avoir une implémentation compilée, et une autre interprétée.

Les techniques de compilation sont très complexes. Pour simplifier la réalisation d'un compilateur, on peut utiliser pour les parties avant et arrière des outils qui simplifient le travail :

- analyse lexicale : lex, flex, etc...
- analyse syntaxique : yacc, bison, etc...
- génération de code (transformation du code intermédiaire en code objet, avec optimisation) : Low Level Virtual Machine, Architecture description language, etc...

Compilateurs particuliers

Compilateur croisé

Un compilateur croisé (en anglais Cross Compiler) est un programme capable de traduire un code source en code objet ayant un environnement d'exécution (architecture matérielle, système d'exploitation) différent de celui où la compilation est effectuée. Ces compilateurs sont principalement utilisés en informatique industrielle.

Byte code **ou** code octet

Certains compilateurs traduisent un langage source en langage machine virtuel, c'est-à-dire en un code (généralement une suite d'octets) exécuté par une machine virtuelle : un programme émulant les principales fonctionnalités d'un ordinateur. Le portage d'un programme ne requiert ainsi que le portage de la machine virtuelle. C'est le cas du compilateur Java, qui traduit du code Java en bytecode Java (code objet). Une machine virtuelle DotNet peut exécuter du bytecode MSIL produit par les langages de Microsoft C#, Visual Basic ou autres.

Autres compilateurs [modifier]

Si la plupart des compilateurs traduisent un code d'un langage de programmation vers un autre, ce n'est pas le cas de tous les compilateurs. Par exemple, le logiciel LaTeX compile un code écrit dans le langage de formatage de texte LaTeX, pour le convertir en un autre langage, par exemple DVI, HTML, PostScript...

Certains compilateurs traduisent, de façon incrémentale ou interactive, le programme source (tapé par l'utilisateur) en du code machine. Par exemple, certaines implémentations de Common Lisp (comme SBCL) traduisent un bout de programme en du code machine (en mémoire).

Les compilateurs Just In Time (juste à temps) traduisent une représentation intermédiaire (souvent du code octet) en du code machine, de manière progressive.

Le problème de l'amorçage (bootstrap)

Les premiers compilateurs étaient écrits directement en langage assembleur, un langage symbolique élémentaire correspondant aux instructions du processeur cible et quelques structures de contrôle légèrement plus évoluées. Ce langage symbolique doit être assemblé (et non compilé) et lié pour obtenir une version exécutable. En raison de sa simplicité, un programme simple suffit à le convertir en instructions machines.

Les compilateurs actuels sont généralement écrits dans le langage qu'ils doivent compiler ; par exemple un compilateur C est écrit en C, SmallTalk en SmallTalk, Lisp en Lisp, etc. Dans la réalisation d'un compilateur, une étape décisive est franchie lorsque le compilateur pour le langage X est suffisamment complet pour se compiler lui-même : il ne dépend alors plus d'un autre langage (fut-ce de l'assembleur) pour être produit.

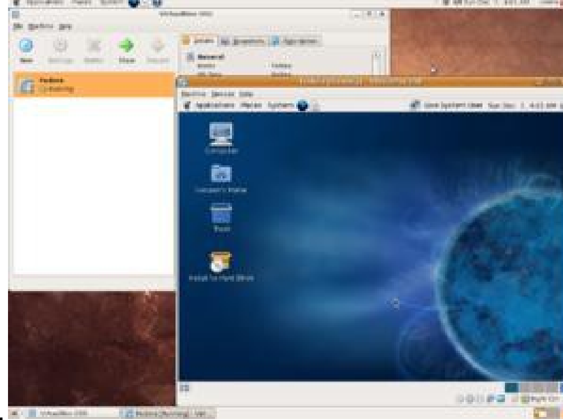
Les bugs des compilateurs sont parfois très complexes à détecter. Si un compilateur de langage C comporte un bug, les programmeurs en langage C auront naturellement tendance à mettre en cause leur propre code source, non pas le compilateur.

Pire, si ce compilateur buggé (version V1) compile un compilateur (version V2) non buggé, l'exécutable compilé (par V1) du compilateur V2 sera buggé. Pourtant son code source est bon.
Le bootstrap oblige donc les programmeurs de compilateurs à contourner les bugs des compilateurs existants.

Machine virtuelle

Le sens originel de **machine virtuelle** (ou Virtual Machine (VM) en anglais) est la création de plusieurs environnements d'exécution sur un seul ordinateur, dont chacun émule l'ordinateur hôte. Cela fournit à chaque utilisateur l'illusion de disposer d'un ordinateur complet alors que chaque machine virtuelle est isolée des autres. Le logiciel hôte qui fournit cette fonctionnalité est souvent dénommé **superviseur** ou **hyperviseur**. Ce concept va plus loin que celui des simples temps

partagés où chaque utilisateur dispose seulement d'un espace de développement personnel, et non d'une machine simulée entière.



- VM/370 a été l'un des premiers systèmes de virtualisation en informatique, et le premier à être diffusé à l'échelle industrielle. Il fonctionnait sur les ordinateurs IBM 370, bien qu'une version destinée aux universités ait fonctionné sur le modèle 67 sous le nom de Control program (CP). Il permettait d'avoir plusieurs systèmes d'exploitation simultanés sur le même mainframe. Pour éviter les duplications inutiles de code, on pouvait définir des segments de mémoire partagés de façon invisible (en mode de lecture seule, bien sûr) entre deux de ces systèmes ou plus.
- Windows NT et ses successeurs incorporent une machine virtuelle pour simuler un environnement MS-DOS; Windows Server 2008 propose un hyperviseur intégré.
- Linux possède lui aussi un environnement de virtualisation Open Source nommé Xen. Par réaction, une version simplifiée du produit précurseur de la société VMware a été rendue gratuite par cet éditeur en 2006.

Historique et principe

L'adoption de la microprogrammation résolvait alors - au prix d'une légère perte d'efficacité - la question des migrations d'une machine à une autre plus puissante, mais le problème de la migration d'un système à un autre plus puissant ne pouvait se résoudre qu'en utilisant deux machines, à une époque où celles-ci étaient onéreuses. Ce problème concernait :

- Migration d'un système à une version ultérieure ;
- Migration du DOS à l'OS ;
- Migration d'un OS à un autre (DOS vers OS, par exemple).

Une solution adoptée par les centres de Cambridge et de Grenoble fut de simuler le comportement d'une machine par une sorte d'application nommée le Control Program. Chaque machines simulée par le Control Program avait son propre système d'exploitation, mais CP déroutait tous les appels vers des commandes directes au matériel (en fait, à des programmes canal : XIO) ou à certaines commandes système pour les simuler.

Intérêt d'une machine virtuelle

Lors de la préparation d'une migration, on peut utiliser simultanément et sans danger pour l'exploitation même en cas de crash système :

- la machine ancienne et la nouvelle qui est simulée dessus, ou l'inverse.
- le système ancien et le nouveau qui est simulé dessus, ou l'inverse.

Sécurité : Les machines virtuelles sont totalement isolées les unes des autres; de plus, en 2006, la plupart des virus testaient immédiatement s'ils tournaient en environnement virtualisé et renonçaient à agir lorsque c'était le cas. Rien ne prouve néanmoins que cette protection ne puisse être contournée d'une manière ou d'une autre, et une littérature abondante[1] est publiée en permanence sur ce sujet.

Facilité d'extension : Le nombre de machines virtuelles se gère quasi indépendamment du nombre de machines réelles, et de façon transparente pour les utilisateurs. Les statistiques de charge des machines virtuelles permettent de les réorganiser sur les machines réelles, ainsi que de prévoir les dates auxquelles prévoir des extensions.

Machine virtuelle émulant des systèmes d'exploitation

Le terme de **machine virtuelle** est aussi depuis quelque temps utilisé dans un sens très différent pour désigner un environnement créé par un émulateur. Celui-ci est un logiciel qui émule un système d'exploitationpour l'utilisateur final. Ce logiciel est une surcouche qui se greffe sur le système d'exploitation natif.

Exemples

- KVM Transforme le noyau Linux en hyperviseur. Cette solution est actuellement incluse dans les version du noyau Linux depuis la 2.6.18.
- Oracle VM
- QEMU
- VMware permet la virtualisation non seulement d'un PC à architecture Intel/AMD (sur lequel on peut alors utiliser simultanément des Windows et des Linux, éventuellement de plusieurs générations différentes), mais de tous les périphériques d'un réseau : un périphérique distant peut apparaître si on le désire comme local !
- VirtualBox Alternative à VMware et autres concurrents qui est passée sous licence GPL. Installable sur différentes architectures (Windows, OS X, Linux, Solaris)[réf. nécessaire], il crée un ordinateur virtuel et permet d'installer n'importe quel type de système d'exploitation, et ses périphériques...
- Virtual PC Gratuit
- Xen
- Bochs

Machine virtuelle parallèle

Plus récemment, le terme de **machine virtuelle** a été utilisé pour désigner une machine virtuelle parallèle (PVM). Dans ce cas, une machine virtuelle crée un environnement qui semble être un seul ordinateur alors que les ressources de plusieurs ordinateurs sont utilisées.

Exemples

- PVM est une machine virtuelle permettant d'exécuter un programme sur plusieurs ordinateurs.

Machine virtuelle applicative

Dans son second sens, maintenant le plus commun, une **machine virtuelle** désigne un logiciel ou interpréteur qui isole l'application utilisée par l'utilisateur des spécificités de l'ordinateur, œst-à-dire de celles de son architecture ou de son système d'exploitation. Cette indirection permet au concepteur d'une application de la rendre disponible sur un grand nombre d'ordinateurs sans les contraintes habituelles à la rédaction d'un logiciel portable tournant directement sur l'ordinateur. La technologie JIT permet dans bien des cas à l'application d'avoir des performances comparables à une application native.

Windows XP et similaires tournent également dans un environnement virtualisé, qui est créé par la couche HAL. En cas de changement de machine physique, on peut en principe ne changer que le HAL sans toucher au reste de son installation Windows.

.NET vs. Java et Java EE

La CLI et C# ont plusieurs similarités avec la JVM de Sun et Java. Les deux sont basés sur une **machine virtuelle** qui cache les détails matériels de l'ordinateur sur lequel leurs programmes s'exécutent. Les deux utilisent leur propre langage intermédiaire bytecode Common Intermediate Language (CIL, anciennement MSIL) pour Microsoft et Java byte-code pour Sun. Avec .NET, le byte-code est toujours compilé avant l'exécution, soit juste-à-temps (JIT), ou en avance en utilisant l'utilitaire ngen.exe. Avec Java, le byte-code est soit interprété, soit compilé en avance ou encore compilé juste-à-temps. Les deux fournissent des bibliothèques de classes extensibles qui résolvent plein de problèmes de programmation courants, et les deux résolvent beaucoup de problèmes de sécurités par la même approche. Les espaces de noms fournis par le .NET Framework ressemblent beaucoup au package de l'API Java EE aussi bien dans le style que dans l'invocation.

.NET dans sa forme complète (à savoir l'implémentation Microsoft) est actuellement disponible entièrement pour Windows et partiellement pour Linux et Mac, alors que Java est entièrement disponible sur presque toutes les plates-formes. Depuis le début, .NET supporte plusieurs langages et demeure indépendant de la plateforme de telle sorte que n'importe qui peut le réimplémenter sur d'autres plates-formes (l'implémentation Microsoft cible uniquement Windows, Windows CE et la Xbox360). La plate-forme Java a été initialement construite pour supporter uniquement le langage Java, mais sur plusieurs systèmes d'exploitations avec le slogan « Write once, run anywhere » (écrit

une fois, tourne n'importe où). D'autres langages ont été développés pour la machine virtuelle java, mais ils ne sont pas très utilisés. L'implémentation Java de Sun est open source (ce qui inclut la bibliothèque de classes, le compilateur, la JVM ainsi que quelques autres outils associés à la plate-forme Java) sous la licence GNU GPL.

RIA-RDA

De plus en plus de technologies se rejoignent actuellement pour fonctionner coté web et OS. Elles se différencient plus par les outils de production, maintenance que par les technologies utilisées qui se résument en l'utilisation d'un langage de compilation et d'éxécution (**machine virtuelle**) / interprétation et un formalisme xml (xaml, xmm, xul). Ces technologies se retrouvent selon les cas sur des ordinateurs, des téléphones, des pdaphones ou des périphériques multimédia (récepteurs satellite).

Les RIA sont basées sur une machine virtuelle fonctionnant dans un navigateur web. Les RDA sont basées sur une machine virtuelle fonctionnant sur le système d'exploitation. Les RIA et RDA en évoluant sont de plus en plus multi-langage (C#, javascript, java,C#), multi-interface (swing, wpf, flash, AJAX-ui-widgets), multi OS (Mac, Window, unix), multi-matériel (PC, pda, téléphone).

technologie web et applicatif

Société	**technologie**	**Système d'exploitation**	**Langage de programmation**/	**application**	**Web**	format
Microsoft	CLI Microsoft	Microsoft .NET	C#puis CLR/DLR	MFC/ GDI	Silverlight	XAML .cs->.dll
Novell	CLI Microsoft	unix/SuSE	C#puis CLR/DLR	Mono	Moonlight	
Adobe	Flash player	Animation flash	Actionscript	AIR	Adobe Flex	.as ->.swf
Sun Microsystems	Java	Applet Java	java	JFC awt-swing	JavaFX	.java->.class
Netscape	navigator	unix	javascript		AJAX XUL SMIL SVG	

Exemples

- La Machine virtuelle Java permet d'exécuter du code binaire Java
- CLR,DLR dans la plateforme .NET
- Projet Tamarin,SpiderMonkey : supporte l'exécution du javascript à l'intérieur d'un navigateur web
- Flash supportant l'execution d'ActionScript
- Parrot
- Low Level Virtual Machine
- Rich Internet Application

▪ Bytecode

Le bytecode est un code intermédiaire plus concret que le code machine non directement exécutable. Il est contenu dans un fichier binaire qui représente un programme, tout comme un fichier objet produit par un compilateur.

Il est appelé bytecode du fait de son format où chaque instruction est codée en binaire.

Puisque c'est un code qui n'est pas exécutable directement par un processeur (à l'exception de certains processeurs gérants le bytecode Java nativement), il est utilisé par les créateurs de langages de programmation comme un code intermédiaire réduisant la dépendance avec le matériel et facilitant son interprétation sur plusieurs architectures.

Certains compilateurs utilisent le bytecode comme représentation intermédiaire. Certains systèmes, appelés « traducteurs dynamiques », ou « compilateurs JIT (just-in-time) », traduisent le bytecode en code machine juste avant l'exécution, afin d'en augmenter la vitesse.

Un programme à base de bytecode est exécuté par un interpréteur appelé machine virtuelle, du fait qu'il s'agit d'un programme qui exécute le code tout comme un microprocesseur. L'avantage est la portabilité : le même bytecode peut être exécuté sur diverses plates-formes ou architectures pour laquelle un interpréteur existe. Un programme sous forme de bytecode peut donc être transmis d'une machine à une autre et être exécuté sans modification par la machine exécutrice quelle qu'elle soit. L'avantage est le même que pour les scripts, qui sont directement interprétés (et non compilés en bytecode). Cependant, le bytecode est plus abstrait, plus compact et plus facile à manipuler qu'un script, prévu, lui, pour être humainement intelligible. De ce fait les performances des interpréteurs de bytecode sont généralement bien meilleures que celles des interpréteurs de scripts.

Pour bénéficier de ces avantages, aujourd'hui de nombreux langages interprétés sont en fait compilés en bytecode avant d'être exécutés par un interpréteur. C'est le cas par exemple de PHP (lorsqu'il est utilisé pour des applications), Tcl et de Python. Un programme Java est habituellement transmis sous forme de bytecode à une machine hôte qui utilisera un compilateur JIT pour traduire le bytecode en code machine avant exécution. Les implémentations actuelles de Perl et de Ruby n'utilisent pas de bytecode, mais une structure en arbre qui se rapproche de la représentation intermédiaire des compilateurs.

Les p-Codes diffèrent des bytecodes par le codage de leurs opérations, qui peut être de plusieurs octets avec une taille variable, tout comme les opcodes de nombreux processeurs. Ils ont un plus haut niveau descriptif, comme « afficher cette chaine de caractères » ou encore « effacer l'écran ». Le BASIC et quelques versions de Pascal utilisent un p-Code.

Exemples

- O-code de BCPL

- CLISP, une implantation de Common Lisp, ne compile qu'en bytecode
- CMUCL, une implantation de Common Lisp, peut compiler en bytecode ou en code natif ; le code natif est bien plus compact
- OCaml peut compiler un bytecode compact
- Microsoft .NET Common Intermediate Language, exécuté par le Common Language Runtime
- Java de Sun Microsystems (voir Bytecode Java).
- Python

Tél : (00229) 96 15 81 18 E-mail : cyze2002@yahoo.fr / cyze2002@hotmail.com