USTHB

Faculté d'Informatique

Département d'Intelligence Artificielle et de Sciences de Données

TP1 (SYSTEMES D'EXPLOITATION –M1:IV)

Concurrence 10/10/2025 Année 25/26

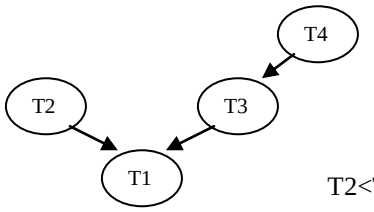
Présentation

Le but de ce TP est de découvrir le parallélisme à travers la création de processus sous UNIX. Cette possibilité offerte par le système se révèle utile lorsqu'on cherche à implanter des algorithmes parallèles. On se propose de générer un graphe de précédences à partir d'une expression arithmétique complètement parenthèsée.

Exemple :

((A + B) * (C - (D / E))) aura pour graphe :

2 1 3 4



T1 : M1 :=M2*M3
T2 : M2 := A+B
T3 : M3 :=C-M4
T4 : M4 := D/E

T2<T1 ; T3<T1 ; T4<T3.

Génération mono processus

La première version du programme *générateur* sera basée sur une analyse descendante de l'expression par un seul processus, par un appel d'une fonction récursive *génère ()*. L'un de ses arguments est la chaîne de caractères définissant la sous expression à traiter. Elle aura donc pour rôle de travailler à chaque fois sur une sous expression comme suit :

- Elle cherche son opérateur central. Dans l'exemple ci-dessus et pour l'expression entière, l'opérateur principal est '*'.
- Puis elle génère le nœud (ou tache) correspondante. Dans l'exemple ci-dessus et pour l'expression entière, le nœud est T1.
- Puis elle appelle la même fonction pour travailler sur la sous expression de gauche si celle-ci n'est pas un simple opérande (variable ou constante).
- Puis elle appelle la même fonction pour travailler sur la sous expression de droite si celle-ci n'est pas un simple opérande (variable ou constante).
- Puis elle génère le contenu de la tache. Dans l'exemple ci-dessus et pour l'expression entière, la tache est : T1 : M1 := M2*M3
- Puis elle génère la précédence avec la tache générée dans la fonction appelante s'il y a lieu. Dans l'exemple ci-dessus et pour l'expression entière, la tache correspondante à l'opération '*' (T1) n'a pas de tache qui la précède. Par contre, lors du traitement concernant l'opération '+', sa tache correspondante (i.e. T2) précède T1 (i.e. T2<T1 est donc générée).

En conclusion, il y a autant d'appels à la fonction *génère ()* que d'opérations dans l'expression.

Génération parallèle

Dans les appels récursifs précédents, les deux appels sur les sous formules sont logiquement indépendants. Ils peuvent donc être réalisés par deux processus différents, ce qui fournit un schéma de parallélisation de notre générateur. Donc, pour réaliser le travail demandé, le processus père crée deux processus fils dont chacun se charge d'une sous expression, attend leurs terminaisons, achève son travail et conclue. Finalement, il y a autant de processus que d'opération dans l'expression.

Travail demandé

Réaliser ce travail pratique en passant par les trois étapes indiquées ci-dessus.

Connaissances requises

- manipulation des commandes essentielles de Linux (création de répertoire, édition de fichier, compilation, exécution, suppression de fichier, listing du contenu de répertoire...)
- langage de programmation C.
- appels systèmes : création de processus, attente de la fin d'un processus.

Pour vos tests, utilisez : (((A+B)*C)-(((D-(F/G))*(H+(K*L)))/((M-N)*O))))

Echéance

28 Octobre 2025.

USTHB

Faculty of Computer Sciences
Department of artificial intelligence and data science
Practical Work N° 01

(Concurrency)

The 10/10/2025 - Year 2025/2026

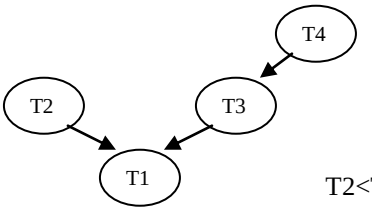
Presentation

The aim of this practical work is to discover parallelism through the creation of processes under UNIX. This possibility offered by the system proves useful when seeking to implement parallel algorithms. We propose to generate a precedence graph from a fully parenthesised arithmetic expression.

Example:

((A + B) * (C - (D / E))) will have as graph :

2 1 3 4



T1 : M1 :=M2*M3
T2 : M2 := A+B
T3 : M3 :=C-M4
T4 : M4 := D/E

T2<T1 ; T3<T1 ; T4<T3.

Single-process generation

The first version of the *generator* program will be based on a top-down analysis of the expression by a single process, by calling a recursive function *generates().* One of its arguments is the character string defining the sub-expression to be processed. Its role will therefore be to work on each sub-expression as follows:

- It searches for its central operator. In the example above and for the entire expression, the main operator is "*".
- Then it generates the corresponding node (or task). In the example above and for the entire expression, the node is T1.
- Then it calls the same function to work on the left sub-expression if the latter is not a simple operand (variable or constant).
- Then it calls the same function to work on the right sub- expression if the latter is not a simple operand (variable or constant).
- Then it generates the task content. In the example above and for the entire expression, the task is::
  T1 : M1 := M2*M3
- Then it generates the precedence with the task generated in the calling function, if there is one. In the example above and for the entire expression, the task corresponding to the "*" operation (i.e. T1) has no preceding task. However, when processing the "+" operation, its corresponding task (i.e. T2) precedes T1 (i.e. T2<T1 is therefore generated).

In conclusion, there are as many calls to the *generate* () function as there are operations in the expression.

Parallel generation

In the previous recursive calls, the two calls on the sub-formulas are logically independent. They can therefore be performed by two different processes, which provides a parallelisation scheme for our generator. So, to perform the requested task, the parent process creates two child processes, each one handles a sub-expression. It then waits for them to finish, completes its task, and concludes. Finally, there are as many processes as there are operations in the expression.

Requested work

Perform this practical task by following the three steps outlined above.

Required knowledge

- Handling essential Linux commands (creating directories, editing files, compiling, executing, deleting files, listing directory contents, etc.)
- C programming language.
- Creating processes, waiting for a process to finish.

For your tests, use: (((A+B)*C)-(((D-(F/G))*(H+(K*L)))/((M-N)*O))))

Deadline

28 October 2025.