

Concurrence

Quentin Fortier

January 29, 2025

Définition

Un programme est une suite d'instructions dans un langage de programmation et stocké dans un fichier appelé code source.

Définition

Un programme est une suite d'instructions dans un langage de programmation et stocké dans un fichier appelé code source.

Définition

Un processus est une instance d'un programme en cours d'exécution. Il est composé d'un espace mémoire, d'un identifiant de processus (PID)...

Processus et thread

Définition

Un programme est une suite d'instructions dans un langage de programmation et stocké dans un fichier appelé code source.

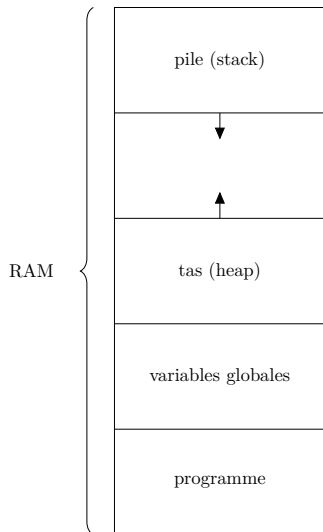
Définition

Un processus est une instance d'un programme en cours d'exécution. Il est composé d'un espace mémoire, d'un identifiant de processus (PID)...

Définition

Un thread (ou fil d'exécution) est une unité d'exécution plus petite qu'un processus. Un processus peut contenir plusieurs threads.

Espace mémoire d'un processus



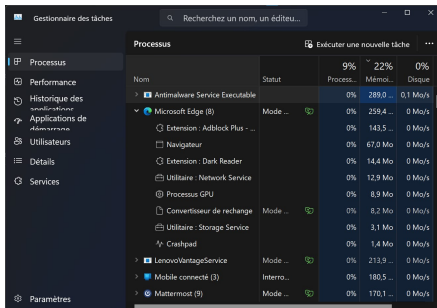
Les threads d'un même processus partagent le même programme, le même tas et les mêmes variables globales.

Par contre, ils ont chacun leur propre pile.

Rappels :

- La pile contient les variables locales, qui sont automatiquement libérées en sortant de leurs portées. La pile est de taille fixe donc limitée (d'où l'erreur `stack overflow`).
- Le tas contient les variables allouées dynamiquement avec `malloc`. L'accès au tas est plus lent que la pile.

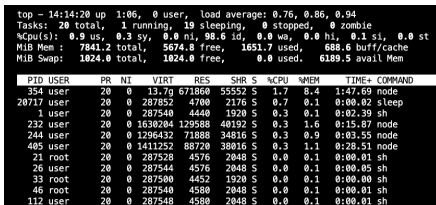
Processeur et cœur



The screenshot shows the Windows Task Manager window with the 'Processus' (Processes) tab selected. The window title is 'Gestionnaire des tâches'. A search bar at the top says 'Recherchez un nom, un éditeur...'. The left sidebar has icons for 'Processus', 'Performance', 'Historique des applications', 'Applications de démarrage', 'Utilisateurs', 'Détails', 'Services', and 'Paramètres'. The main area lists running processes with columns for 'Nom' (Name), 'Statut' (Status), and resource usage (9%, 22%, 0% for Process, Memory, and Disk respectively). The processes listed include 'Antimalware Service Executable', 'Microsoft Edge (8)', 'Extension : AdBlock Plus - ...', 'Navigateur', 'Extension : Dark Reader', 'Utilitaire : Network Service', 'Processus GPU', 'Convertisseur de rechange', 'Utilitaire : Storage Service', 'Crashpad', 'LenovoVantageService', 'Mobile connecté (3)', and 'Mattermost (9)'.

Nom	Statut	9%	22%	0%
		Process...	Mémoi...	Disque
Antimalware Service Executable		0%	289.0 ...	0.1 Mo/s
Microsoft Edge (8)	Mode ...	0%	259.4 ...	0 Mo/s
Extension : AdBlock Plus - ...		0%	143.5 ...	0 Mo/s
Navigateur		0%	67.0 Mo	0 Mo/s
Extension : Dark Reader		0%	14.4 Mo	0 Mo/s
Utilitaire : Network Service		0%	12.9 Mo	0 Mo/s
Processus GPU		0%	8.9 Mo	0 Mo/s
Convertisseur de rechange	Mode ...	0%	8.2 Mo	0 Mo/s
Utilitaire : Storage Service		0%	3.1 Mo	0 Mo/s
Crashpad		0%	1.4 Mo	0 Mo/s
LenovoVantageService	Mode ...	0%	213.9 ...	0 Mo/s
Mobile connecté (3)	Interro...	0%	180.5 ...	0 Mo/s
Mattermost (9)	Mode ...	0%	170.1 ...	0 Mo/s

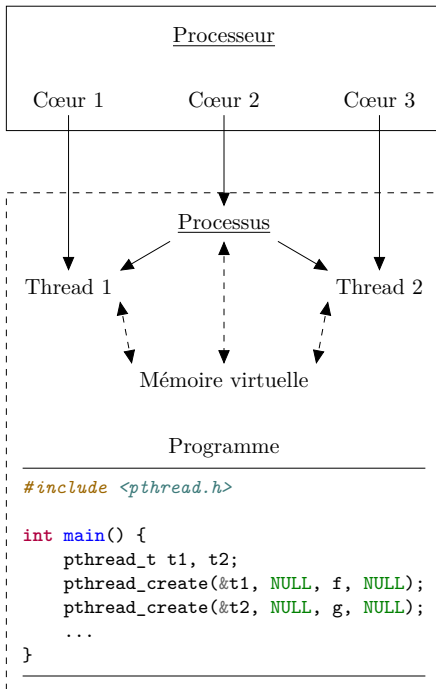
Gestionnaire des tâches de Windows



The screenshot shows a Linux terminal window. The top part displays system statistics: 'top - 14:14:20 up 1:06, 0 user, load average: 0.76, 0.86, 0.94'. Below this, it shows task counts: 'Tasks: 20 total, 1 running, 19 sleeping, 0 stopped, 0 zombie'. Then, CPU usage: '%Cpu(s): 0.9 us, 0.3 sy, 0.0 ni, 98.6 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st'. Memory usage: 'MiB Mem: 7841.2 total, 5674.8 free, 1651.7 used, 688.6 buff/cache'. Swap usage: 'MiB Swap: 1024.0 total, 1024.0 free, 0.0 used, 6189.5 avail Mem'. The bottom part shows the output of the 'top' command, which lists running processes with columns for PID, USER, PR, NI, VIRT, RES, SHR, S, %CPU, %MEM, TIME+, and COMMAND.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
354	user	20	0	13.7g	671860	55552	S	1.7	8.4	1:47.69	node
20717	user	20	0	287852	4700	2176	S	0.7	0.1	0:00.02	sleep
1	user	20	0	287540	4440	1920	S	0.3	0.1	0:02.39	sh
232	user	20	0	1630204	129588	40192	S	0.3	1.6	0:15.87	node
244	user	20	0	1296432	71888	34816	S	0.3	0.9	0:03.55	node
405	user	20	0	1411252	88720	38016	S	0.3	1.1	0:28.51	node
21	root	20	0	287528	4576	2048	S	0.0	0.1	0:00.01	sh
26	user	20	0	287544	4576	2048	S	0.0	0.1	0:00.05	sh
33	root	20	0	287500	4452	1920	S	0.0	0.1	0:00.00	sh
46	root	20	0	287540	4580	2048	S	0.0	0.1	0:00.01	sh
112	user	20	0	287548	4580	2048	S	0.0	0.1	0:00.01	sh

Commande top sous Linux/macOS



Définition

Programmation parallèle : un programme exécute des threads sur plusieurs processeurs (ou cœurs) en même temps.

Intérêt : accélérer l'exécution d'un programme.

Définition

Programmation parallèle : un programme exécute des threads sur plusieurs processeurs (ou cœurs) en même temps.

Intérêt : accélérer l'exécution d'un programme.

Même avec un seul cœur, on peut simuler entrelacer les instructions des différents processus ou threads.

Exemple : gestion des applications sur un système d'exploitation. Les applications sont exécutées en alternance.

Définition

Programmation parallèle : un programme exécute des threads sur plusieurs processeurs (ou cœurs) en même temps.

Intérêt : accélérer l'exécution d'un programme.

Même avec un seul cœur, on peut simuler entrelacer les instructions des différents processus ou threads.

Exemple : gestion des applications sur un système d'exploitation. Les applications sont exécutées en alternance.

Définition

Programmation concurrente : un programme exécute des threads en même temps ou en alternance.

Threads en C

```
#include <pthread.h> // threads POSIX (standard Linux)

void* f(void* x) {
    int* n = (int*)x; // Conversion du type
    for(int i = 0; i < 100000; i++)
        if(i % 20000 == 0)
            printf("%d %d\n", *n, i);
}

int main() {
    pthread_t t1, t2;
    int n1 = 1, n2 = 2;
    pthread_create(&t1, NULL, f, (void*)&n1);
    pthread_create(&t2, NULL, f, (void*)&n2);
    pthread_join(t1, NULL); // Attendre la fin de t1
    pthread_join(t2, NULL); // Attendre la fin de t2
}
```

Compilation : gcc -pthread exemple.c

`void*` est un pointeur (adresse) vers un type quelconque (inconnu). Il permet du polymorphisme en C, comme en OCaml où une fonction peut avoir un type générique `'a` en argument.

Threads en OCaml

```
let f x = ...  
let t = Thread.create f x  
Thread.join t (* Attendre la fin de t* )
```

Threads en OCaml

Exemple concret :

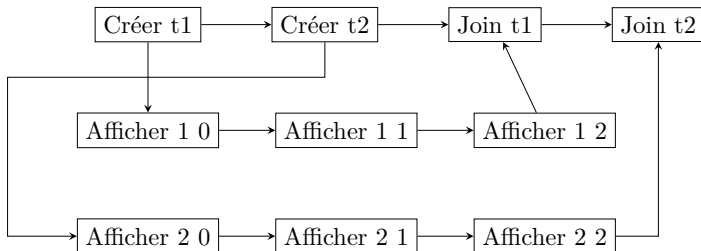
```
let f x =  
  Printf.printf "Thread %d\n" x;  
  for i = 0 to 2 do  
    Printf.printf "%d %d\n" x i  
  done  
  
let () =  
  let t1 = Thread.create f 1 in  
  let t2 = Thread.create f 2 in  
  Thread.join t1;  
  Thread.join t2
```

Compilation :

```
ocamlopt -I +unix -I +threads unix.cmxa threads.cmxa  
exemple.ml
```

Graphe des exécutions

On peut représenter les exécutions possibles du programme précédent par un graphe, où un arc $u \rightarrow v$ signifie que v est exécuté après u .



Graphe des exécutions

Dans un programme séquentiel, on a un ordre total des instructions.

Graphe des exécutions

Dans un programme séquentiel, on a un ordre total des instructions.

Dans un programme concurrent, on a seulement un ordre partiel, défini par la relation d'accessibilité du graphe précédent : s'il y a un chemin de u à v , alors u doit être exécuté avant v .

Définition

Une trace d'un programme est l'ordre dans lequel les instructions sont exécutées (qui respecte le graphe précédent).

Une trace possible :

Créer t1	Afficher 1 0	Créer t2	Afficher 2 0	Afficher 2 1	Afficher 1 1	...
----------	--------------	----------	--------------	--------------	--------------	-----

Exercice

On considère une variable n initialisée à 0 et trois fils d'exécutions qui effectuent les opérations suivantes :

- $T_1 : a \leftarrow n, n \leftarrow 1, b \leftarrow n ;$
- $T_2 : c \leftarrow n, n \leftarrow 2, d \leftarrow n ;$
- $T_3 : e \leftarrow n, f \leftarrow n.$

On suppose que l'instruction $x \leftarrow y$ consiste à écrire le contenu de y dans x et est une instruction atomique. Après l'exécution des trois fils :

- 1 que peut valoir c ?
- 2 que peut valoir b ?
- 3 que peut valoir e ?
- 4 si c vaut 1, que peut valoir d ?
- 5 si f vaut 0, que peut valoir e ?

Embarassingly parallel

Le parallélisme est particulièrement simple quand les calculs peuvent être faits de manière indépendante.

Exercice

Décrire un programme calculant le produit de deux matrices de manière parallèle. Quel est le gain de temps par rapport à une version séquentielle ?

Compteur et opération atomique

```
int counter;
void* increment(void* arg){
    for (int i = 1; i <= 1000000; i++) {
        counter++;
    }
}
int main(){
    counter = 0;
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}
```

Question

Quelle est la valeur de counter à la fin de l'exécution de main ?

Compteur et opération atomique

```
int counter;
void* increment(void* arg){
    for (int i = 1; i <= 1000000; i++) {
        counter++;
    }
}
int main(){
    counter = 0;
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}
```

Question

Quelle est la valeur de counter à la fin de l'exécution de main ?

Pas forcément 2000000...

Définition

Une opération est atomique si elle est exécutée en une seule fois, sans être interrompue.

Définition

Une opération est atomique si elle est exécutée en une seule fois, sans être interrompue.

Une opération élémentaire (lecture ou écriture d'une variable de type `int` par exemple) est atomique.

Définition

Une opération est atomique si elle est exécutée en une seule fois, sans être interrompue.

Une opération élémentaire (lecture ou écriture d'une variable de type `int` par exemple) est atomique.

`counter++` n'est pas atomique, car elle est traduite en assembleur par :

```
int tmp = counter;  
tmp = tmp + 1;  
counter = tmp;
```

```
movl    counter(%rip), %eax  
addl    $1, %eax  
movl    %eax, counter(%rip)
```

Compteur et opération atomique

```
int counter;
void* increment(void* arg){
    for (int i = 1; i <= 1000000; i++) {
        int tmp = counter;
        tmp = tmp + 1;
        counter = tmp;
    }
}

int main(){
    counter = 0;
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}
```

La valeur maximum de counter est

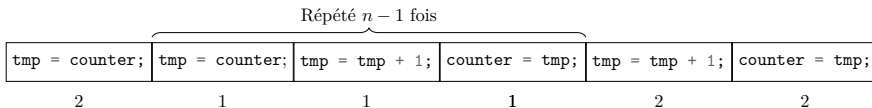
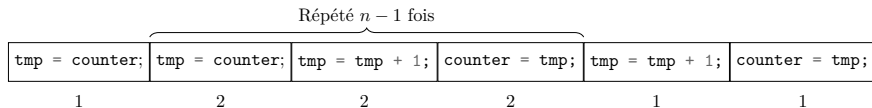
La valeur maximum de counter est 2000000.

La valeur minimum de counter est

La valeur maximum de counter est 2000000.

La valeur minimum de counter est 2 :

- le thread 1 lit 0 dans counter
- le thread 2 fait $n - 1$ itérations
- le thread 1 écrit 1 dans counter
- le thread 2 lit 1 dans counter
- le thread 1 fait $n - 1$ itérations
- le thread 2 écrit 2 dans counter



Exercice

Quelle est la valeur minimum et maximum de `counter` si on utilise k threads ?

Compteur et opération atomique

Remarque : Il existe des types et structures de données qui garantissent des opérations atomiques, comme `std::atomic<int> counter;` en C++ et `Atomic` en OCaml.

Mais cela ne permet pas de définir des sections critiques.