Profesor: Dani Jimenez

Serial vs Parallel

Serial

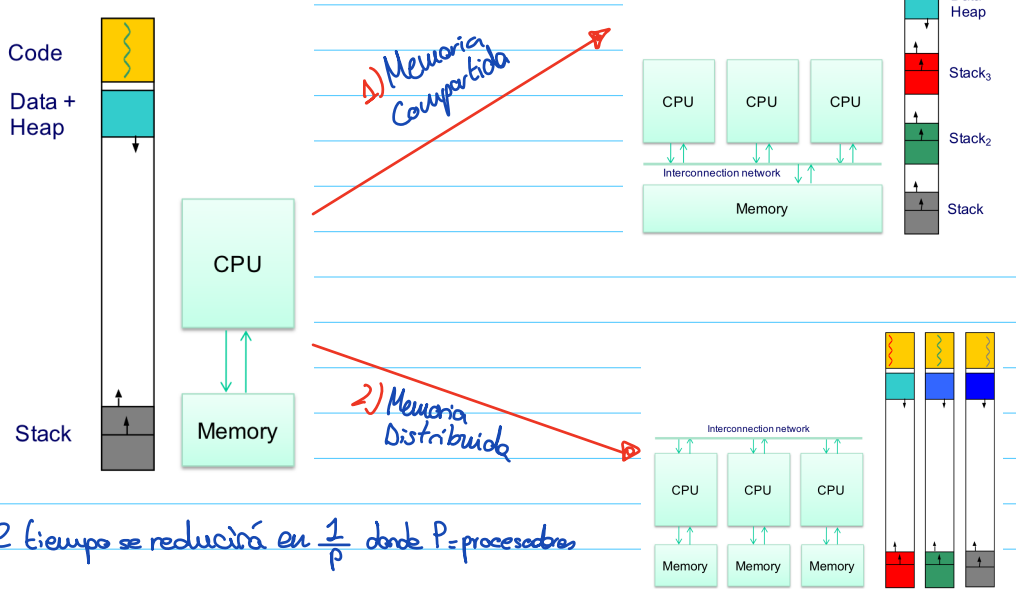
En una ejecución serial, el tiempo de ejecución viene dado por:

N = nº instr
$T = N/F$
$F = IPC \cdot f$
↳ Inst. por ciclo

→ Para reducir tiempo: ↑ F

Parallel

Otra manera de reducir el tiempo es mediante arquitecturas paralelas, pueden ser:



El tiempo se reducirá en $\frac{1}{P}$ donde P = procesadores

Nos queda entonces que:

$T = (N/P)/F$
N = nº instr
P = nº procesadores
F = Frecuencia

nos centraremos en esta

⚠ Habrá que coordinar las tareas (trozos de la aplicación) para un correcto acceso

Throughput computing

También podemos explotar la arquitectura paralela corriendo diferentes aplicaciones en diferentes procesadores a la vez [definición throughput computing]

K programas y P procesadores ⟶ cada programa recibe $\frac{P}{K}$ procesadores

El SO multiplexa su ejecución

Concurrencia

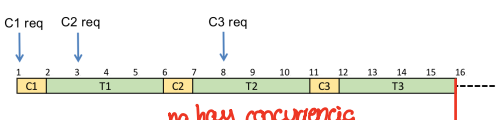
Ejecución concurrente

> Consiste en romper un problema en partes (tareas) para asegurar su ejecución simultánea

Ejemplos

1) Serial ⟶ 1 CPU

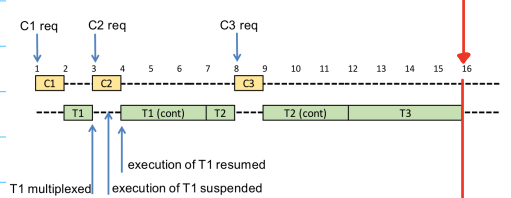
Task Ck: receives client requests (duration 1 time unit)
Task Tk: executes a single bank transaction (e.g. withdraw/deposit some money in bank account, with a duration of 4 time units)

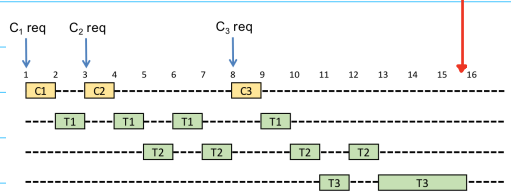


no hay concurrencia

⚠ No tienen por qué ser en orden las tareas

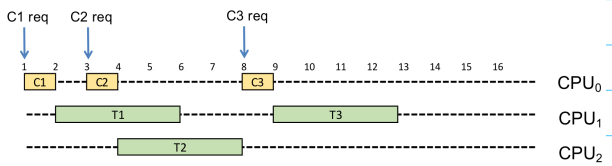
2) Concurrente ⟶ 1 CPU





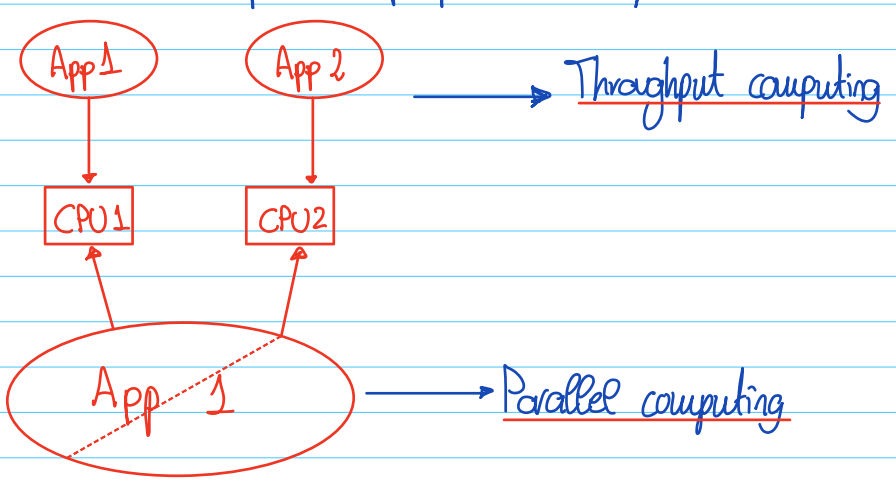
1 CPU, mismo tiempo siempre

3) Paralelo → 3 CPU



Ejecución parallela: Reducir el tiempo de ejecución (response) de un programa.
parallel computing Se usan múltiples CPU para ejecutar tareas identificadas para concurrencia
1 programa → P procesadores
Cada CPU recibe 1/p del programa, se reduce el tiempo en P

Throughput computing: Se usan multiples procesadores para incrementar el número de programas por unidad de tiempo
n programas en P procesadores
→ n > p: cada programa recibe P/n procesos
n ≤ p: cada programa recibe 1 proceso



Problemas Concurrencia

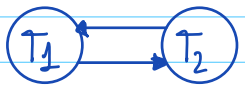
Existen diferentes problemas con la concurrencia:

- **Database condition**

recursos compartidos con dos o más tareas que modifican (leen y escriben)
→ suma = suma + 1
  - Si tenemos 3 tareas que acceden a la vez, suma = 5 no 6!! se solapan

- **Deadlock**

  Dos o más tareas que se bloquean por que se espera a que otra haga algo y esta otra también y así sucesivamente. Bloqueo



- **Livelock**

  Dos más tareas cambian continuamente de estado en respuesta a cambios de otras tareas sin hacer nada útil

- **Starvation**

  Una tarea no puede ganar acceso a un recurso compartido porque está esperando indefinidamente

- **Performance**

Problemas

1. Assume we want to execute two different applications in our parallel machine with 4 processors: application *App1* is sequential; *App2* is parallelised defining 4 tasks, each task executing one fourth of the total application. The sequential time for the applications is 8 and 40 time units, respectively. Assuming: that *App1* starts its execution at time 4 and *App2* starts at time 0, draw a time line showing how they will be executed if the operating system:

   (a) Does not allow multiprogramming, i.e. only one application can be executed at the same time in the system. ⟶ entiendo que solo un CPU no concurrente

   (b) Allows multiprogramming so that the system tries to have both applications running concurrently, each application making use of the number of processors is able to use.

   (c) The same as in the second case, but now *App2* is parallelised defining 3 tasks, each task executing one third of the total application.

4 CPU

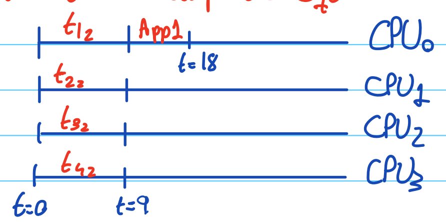
App 1 ⟶ Secuencial ⟶ 8 u.t ⟶ empieza en t=4

App2 ⟶ [t_{1_2} | t_{2_2} | t_{3_2} | t_{4_2}] (0 10 20 30 40 u.t) ⟶ 40 u.t ⟶ empieza en t=0

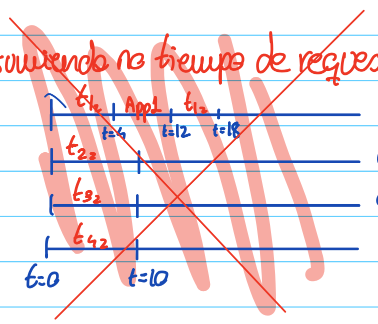
↓

Cada tarea 10 u.t

a) Asumiendo no tiempo de request

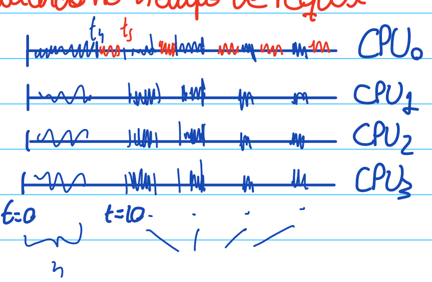


b) Asumiendo no tiempo de request



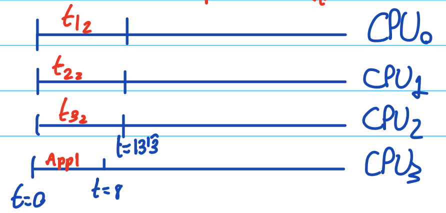
el SO permite que se ej. quantum 1 u.t ⟶ lo superponemos puede ser cualquier

b) Asumiendo no tiempo de request



el throughput computing mejor

c) Asumiendo no tiempo de request



2. Assume we want to execute two different applications (*app1* and *app2*) in our parallel machine with p processors. Both applications can be (ideally) parallelised defining up to p tasks, each task executing one over p of the total application. The sequential time for *app1* and *app2* is 1200 and 2000 time units, respectively. Assuming that the operating system allows the multiprogrammed execution of parallel applications, for $p = 8$ decide the best allocation of processors to both applications in order to minimise the time the user has to wait for them to finish and giving the programmer the impression that both applications are running from the beginning.

App 1 y App 2 ⟶ p tareas ⟶ $\frac{1}{p}$ del total

p procesadores

A_1: 1200 u.t

A_2: 2000 u.t

$p = 8$

} buscar que corran desde el principio ⟶ { A_1: 8 tareas de 150 u.t; A_2: 8 tareas de 250 u.t

No asumo requests, hay que hacerlo?