Introducció (molt breu) a la concurrència

Lluís Garrido

lluis.garrido@ub.edu

Grau d'Enginyeria Informàtica

Concurrència

A la realitat, en el món dels ordinadors, moltes activitats tenen lloc al mateix temps

- Múltiples usuaris poden accedir a diverses pàgines web al mateix temps.
- Un navegador pot carregar diverses pàgines web al mateix temps.
- Les aplicacions amb interfície gràfica mostren el menú mentre fan altres coses.
- Etc...

Concurrència

Com a programadors estem acostumats a pensar de forma "seqüencial". En particular, pensem que

- Un programa comença per la funció principal (en C el main).
- El programa s'executa de forma seqüencial, seguint les instruccions que hi hem indicat fins que acaba.

Es possible que a la nostra aplicació s'executin diverses parts del codi al mateix temps?

Concurrència

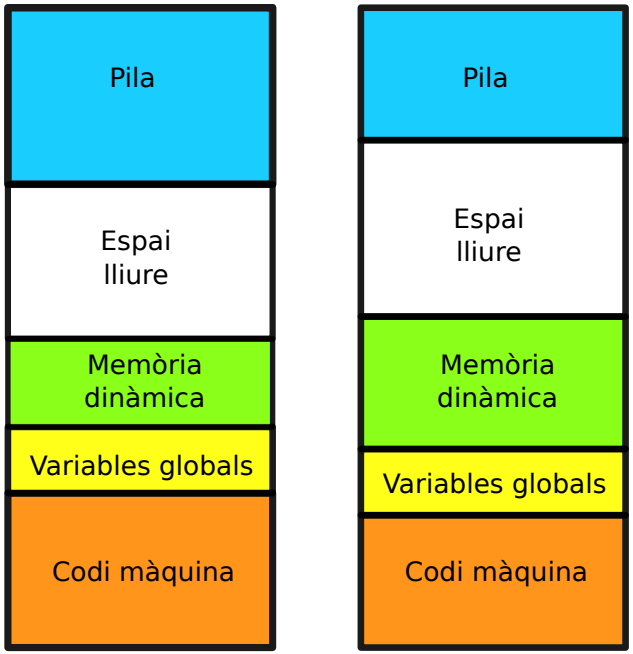
Podem executar diverses parts del codi "al mateix temps"?

- Hem vist els principis bàsics amb la funció `fork`. Processos independents es comuniquen entre sí per repartir-se la feina.
- Actualment el sistema operatiu ens ofereix també eines perquè, en un determinat procés, es puguin executar diverses parts del codi al mateix temps: associat hi ha el concepte de fil (thread, en anglès).

El sistema operatiu ens ofereix eines que permeten gestionar la concurrència, sigui amb múltiples processos o múltiples fils.

Concurrència amb processos

- Cada procés té un espai de memòria independent. Els processos no poden, per defecte, escriure a l'espai de memòria d'un altre procés.
- Per comunicar-se entre sí cal fer servir els serveis dels sistema operatiu. Per exemple, escriure a un fitxer compartit o comunicar-se via una canonada o xarxa.



Concurrència amb fils

Què és un fil? Intuïtivament...

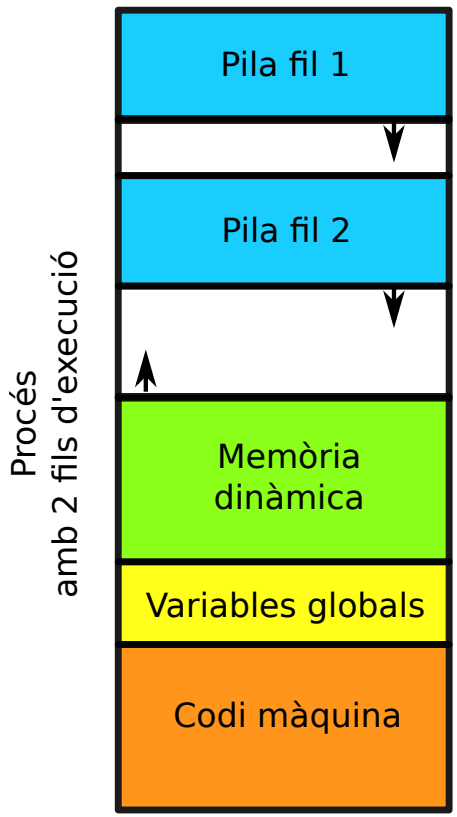
- Un procés està format pel codi (executable) i les variables associades. És com si fos un "llibre d'instruccions".
- Un fil d'execució és el que executa les instruccions del procés i canvia els valors de les variables. El fil és el que llegeix i executa el que posa al "llibre d'instruccions".

Observar que...

- Estem acostumats a pensar que un procés té un únic fil d'execució. És la forma més senzilla de pensar-hi: un programa s'executa de forma seqüencial.
- Però un procés pot tenir múltiples fils d'execució. Parts diferents del codi (o inclús el mateix codi) poden ser executats al mateix temps per fils diferents. Per què funciona?

Concurrència amb fils

- Els fils d'un procés comparteixen l'espai de memòria del procés. Cada fil té la seva pròpia pila i registres de la CPU i pot executar doncs una part diferent de l'aplicació.
- Per comunicar-se entre sí poden fer servir l'espai de memòria que comparteixen.




```
int global;

void funcio()
{
  int a;

  // Aixo ho executen els dos fils
}

void main(...)
{
   int fil;

   // Aixo ho executa un sol fil, una sola pila

   fil = crear_fil_nou(&funcio); // Retorna id fil creat

   // Aixo ho executa un sol fil, hi ha dues piles

   funcio();

   esperar_que_acabi_fil(fil);
}
```

Concurrència amb fils

- Elements d'un procés
  - Espai d'adreces (codi, variables globals, ...)
  - Fitxers oberts
  - La llista de fils del procés
  - Altres
- Elements propis de cada fil
  - Comptador de programa
  - Registres de la CPU
  - Pila
  - Estat (preparat, bloquejat, execució)

Concurrència amb fils

A l'actualitat

- El sistema operatiu realitza (realment) la planificació amb els fils. Cada fil té un estat diferent i el planificador fa el canvi de context a nivell de fil[1].
- Cada fil pot estar executant una part diferent del codi del procès. Inclús el mateix codi pot ser executat per diferents fils a la vegada.

[1] Al tema de planificació es parlava de planificar una "tasca" en comptes de un "procés". Això és per què en el fons es planifiquen fils i no processos.

Concurrència amb fils

Avantatges de la programació mutifil

- Cada fil té el seu propi estat (bloquejat, preparat, execució). Es pot millorar doncs el temps de resposta de l'aplicació.
- Simplificació del codi per tractar amb events asíncrons (ratolí, teclat, xarxa, ...)
- Facilitat per comunicar els fils entre sí (respecte fer-ho amb processos).

Atenció!

- No associar programació multifil amb sistemes multiprocessadors.
- Les avantatges de la programació multifil són evidents també amb sistemes uniprocessador.

Discussió: processos vs. fils

Ens els processos

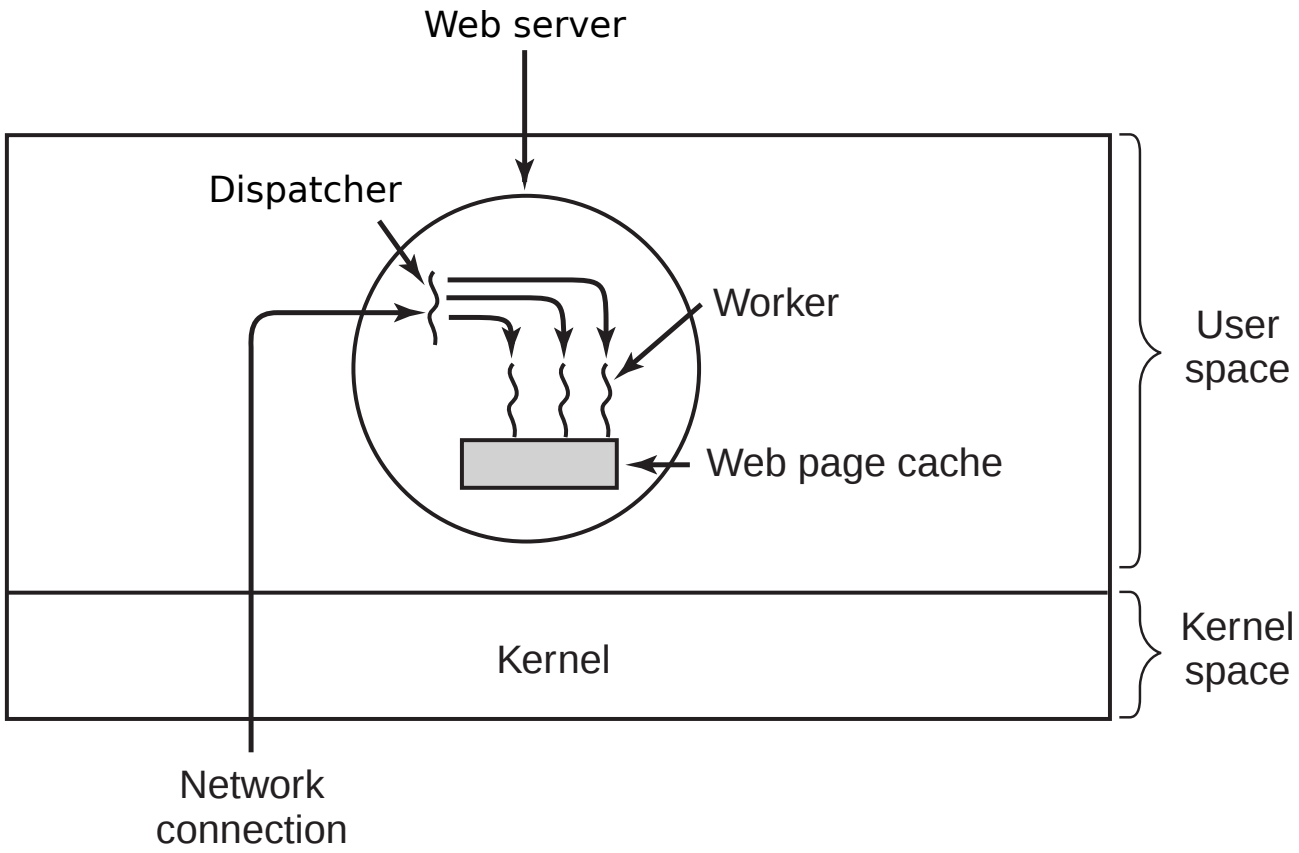
- La comunicació pot ser entre processos executant al mateix o diferents ordinadors.
- La comunicació es realitza mitjançant serveis que ofereix el sistema operatiu.

En els fils

- Tots els fils d'un procés executen al mateix ordinador.
- La comunicació es realitza a través de l'espai de memòria del procés, per exemple, variables globals. No cal utilitzar els serveis que ofereix el sistema operatiu.
- La complexitat se centra en la sincronització i coordinació dels fils perquè es comuniquin correctament entre sí... això ho veurem a Sistemes Operatius 2.

Discussió: processos vs. fils

- Volem dissenyar una aplicació, per exemple un servidor web.
- Quin disseny és millor: múltiples processos o múltiples fils ?



Discussió: processos vs. fils

La decisió depèn de la quantitat de dades a compartir:

- Utilitzar fils si hi ha moltes dades a compartir
- Utilitzar processos si hi ha poques dades a compartir.

O la seguretat que volem implementar:

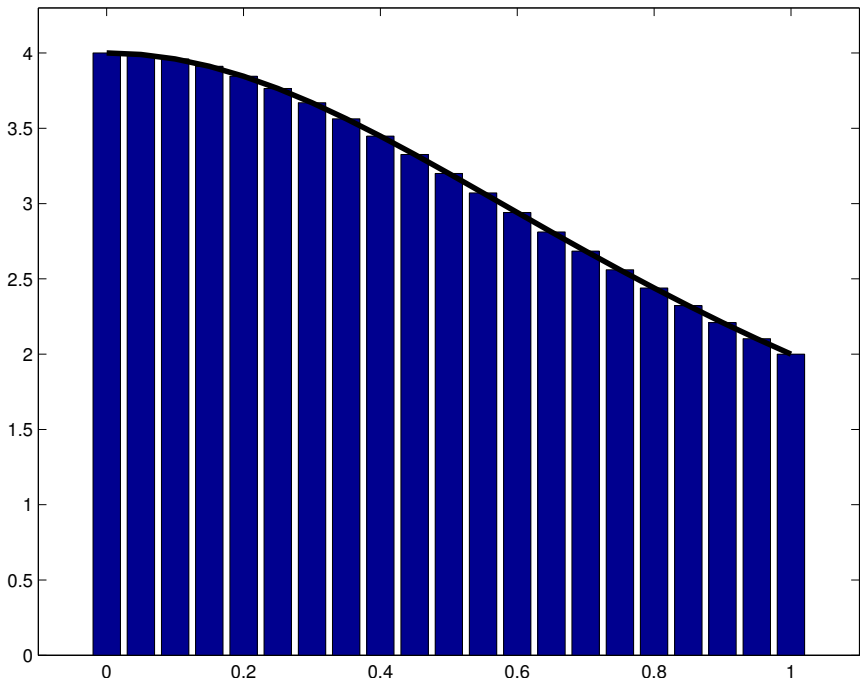
- Si un fil fa una operació invàlida el sistema operatiu mata tot el procés (amb tots els seus fils).
- Si un procés fa una operació invàlida a memòria el sistema operatiu mata el procés que ha fet l'operació invàlida, però no la resta.

Utilitzar processos és adequat doncs en aplicacions crítiques. A Google Chrome cada tabulador és un procés. A Firefox cada tabulador és un fil.

Exemple: processos vs. fils

Recordem l'exemple del càlcul del valor de π

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$




```c
#define NUM_RECTS 100000000
int main()
{
  int i;
  double mid, height, width, sum = 0.0;
  double area;

  width = 1.0 / (double) NUM_RECTS;
  for(i = 0; i < NUM_RECTS; i++) {
    mid = (i + 0.5) * width;
    height = 4.0 / (1.0 + mid * mid);
    sum += height;
  }
  area  = width * sum;
  printf("pi = %e\n", area);

  return 0;
}
```

El càlcul de l'àrea de cada rectangle és independent de la resta.

Exemple: processos vs. fils

- Implementació fent servir 1 procés amb 1 fil. Codi `calcul_pi_1proces.c`.

  ```
  $ time calcul_pi_1proces
  ```

- Implementació fent servir 2 processos, cada procés amb 1 fil. Codi `calcul_pi_2processos.c`.

  ```
  $ time calcul_pi_2processos
  ```

- Implementació fent servir 1 procés amb 3 fils (un és el fil principal, i dos fils més que realizen els càlculs). Codi `calcul_pi_fils.c`

  ```
  $ time calcul_pi_fils
  ```

Exemple: processos vs. fils

A l'exemple anterior

- A l'exemple amb múltiples processos cal fer servir els mecanismes de comunicació que ens ofereix el sistema operatiu. En concret, en aquest exemple es fa servir una canonada perquè comunicar els processos entre sí.
- A l'exemple amb múltiples fils la comunicació es realitza a través d'una variable global: els dos fils que realitzen els càlculs "comuniquen" el resultat parcial al fil principal a través d'una variable global.

Els detalls de la programació multifil els veurem a Sistemes Operatius 2...