Centro de
Investigación en
Computación

Corrutines

Corrutinas

Course

Operating System (with focus on Security)

Instructor

Acosta Bermejo Raúl et al.

Lecture notes

2024-B
7 de noviembre del 2024
Última actualizacioón

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

Introduction

Bibliografía

Los de cabecera:

1. El entorno de programación de UNIX, Brian Kernighan, Prentice Hall.
2. Advanced Unix programming, Marc Rochkind, Prentice Hall.
3. Unix programación avanzada, Francisco Marquez, Adison Wesley Iberoamericana. Didáctico y bueno para programar.

Muy completos aunque no siempre muy didácticos:

- Advanced Programming in the UNIX Environment, W. Richard Stevens.
- UNIX network Programming, APIs: Sockets and XTI, W. Richard Stevens.

Motivación

Problemas

Problemas No paralelizables

Función de Ackerman

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m-1, A(m,n-1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

La serie de Fibonacci

$$f_0 = 0,\ f_1 = 1,\ f_n = f_{n-1} + f_{n-2}$$

Problemas Paralelizables

- Juegos
  - Cada hilo controla el movimiento de un objeto.
- Simulacions científicas
  - Hurricane movement simulation: each thread simulates the hurricane in a small domain.
  - Dinámica molecular: cada hilo simula un subconjunto de partículas.
- Servidor Web
  - Cada hilo atiende una conexión

Motivación

Teoria general

Speedup

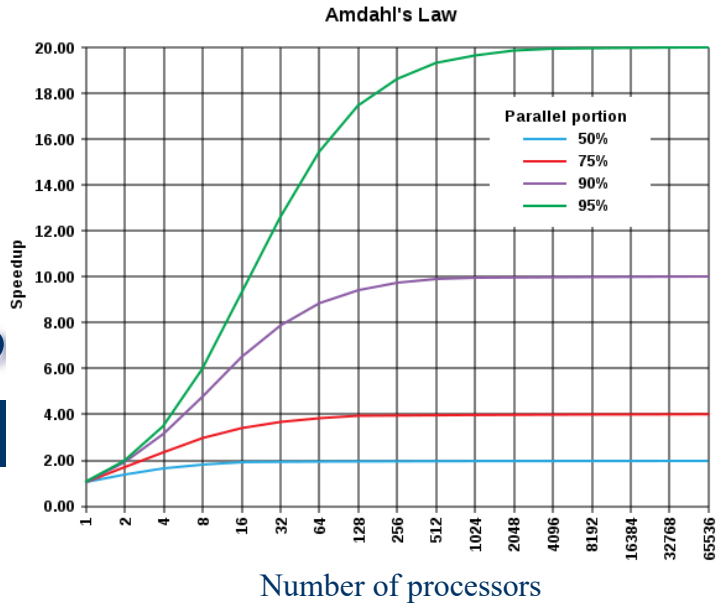
- It is a process for increasing the performance between two systems processing the same problem.
- More technically, it is the improvement in speed of execution of a task executed on two similar architectures with different resources.
- Speedup can be defined for two different types of quantities: *latency* and *throughput*.
- Speedup in latency can be predicted from Amdahl's law or Gustafson's law.

Links

- https://en.wikipedia.org/wiki/Speedup

Motivación

Leyes generales de comportamiento



Amdahl's law (or Amdahl's argument)

- It gives the theoretical **speedup in latency** of the execution of a task at **fixed workload** that can be expected of a system whose resources are improved.
- It is named after computer scientist Gene Amdahl, and was presented at the AFIPS Spring Joint Computer Conference in 1967.
- Amdahl's law can be formulated the following way:

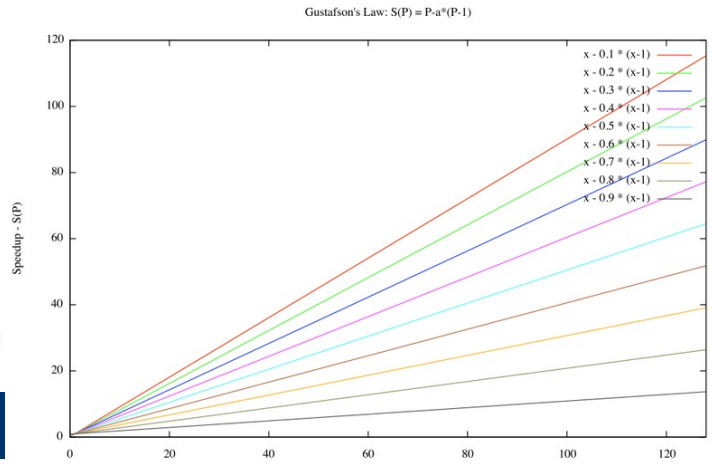
$$S_{latency}(s) = {}^{1}\Big/_{(1-p)+\frac{p}{s}}$$

Workload
Carga de trabajo

- where:
  - $S_{latency}$ is the theoretical speedup in latency of the execution of the whole task.
  - s is the speedup in latency of the execution of the part of the task that benefits from the improvement of the resources of the system.
  - p is the percentage of the execution **time** of the whole task concerning the part that benefits from the improvement of the resources of the system before the improvement.

Motivación

Leyes generales de comportamiento



Gustafson's law (or Gustafson–Barsis's law)

- It gives the theoretical **speedup in latency** of the execution of a task at **fixed execution time** that can be expected of a system whose resources are improved.
- It is named after computer scientist John L. Gustafson and his colleague Edwin H. Barsis, and was presented in the article Reevaluating Amdahl's Law in 1988.
- Gustafson's law can be formulated the following way:

$$S_{latency}(s) = 1 - p + sp$$

- Where: similar to Amdahl's law except for p (workload, here time).
- Other links

  http://www.lnds.net/blog/2009/09/el-problema-de-paralelizar-2.html

Introduction

Entidades concurrentes o paralelas

La evolución de estas entidades está enmarcada por los siguentes conceptos:

1. Corrutinas
2. Procesos
3. Hilos (Preemtivos)
4. Hilos (Cooperativos) o Fibers

Además tambien se han creado los conceptos de:

- Scheduler Preemtivos vs Cooperativos, Tiempo Real
- Evolución: Chunks, Corrutina, Hilo.

Introduction

Nota histórica

Han existido varios lenguajes (como pascal) que introducen instrucciones (palabras reservadas) especiales para crear concurrencia.

CoBegin
  Codigo 1
CoEnd

CoPar
  Codigo 2
Coend

Paralelismo vs Concurrencia

Entidades Concurrentes

Conceptos

Entidades Concurrentes

Conceptos

Paralelismo

- Es el uso de varias unidades de procesamiento (procesadores, núcleos, CPUs, GPUs, etc.) para resolver un problema.
  - El paralelismo se puede dar de manera local o remota, el tiempo de comunicación es el elemento.
    - Cluster Computing: Sistema homogéneo, mismos equipos interconectados localmente (rápidamente) mediante un bus.
    - Grid Computing: Computadoras heterogéneas, interconectadas mediante una red de computadoras. Sistemas Distribuidas.
- Se ejecutan tareas de manera simultanea, al mismo tiempo, en paralelo.
- El paralelismo se puede dar a diferentes niveles, con diferentes granularidades, a nivel de instrucción, de tarea, de operación:
  - Procesadores vectoriales.

URLs

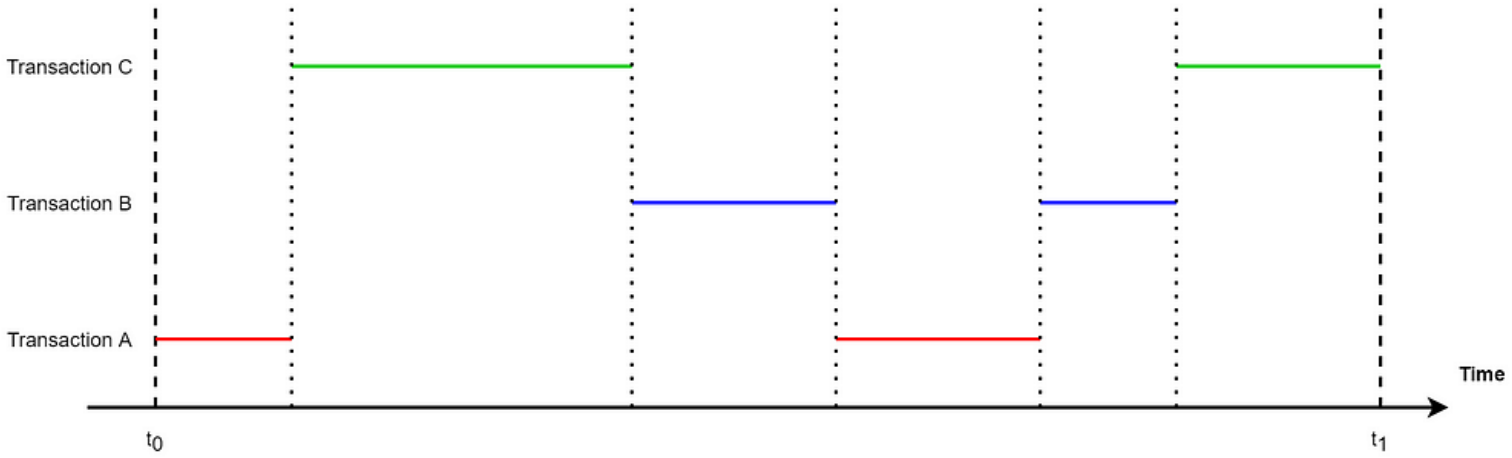
- https://es.wikipedia.org/wiki/Computación_paralela

Entidades Concurrentes

Conceptos

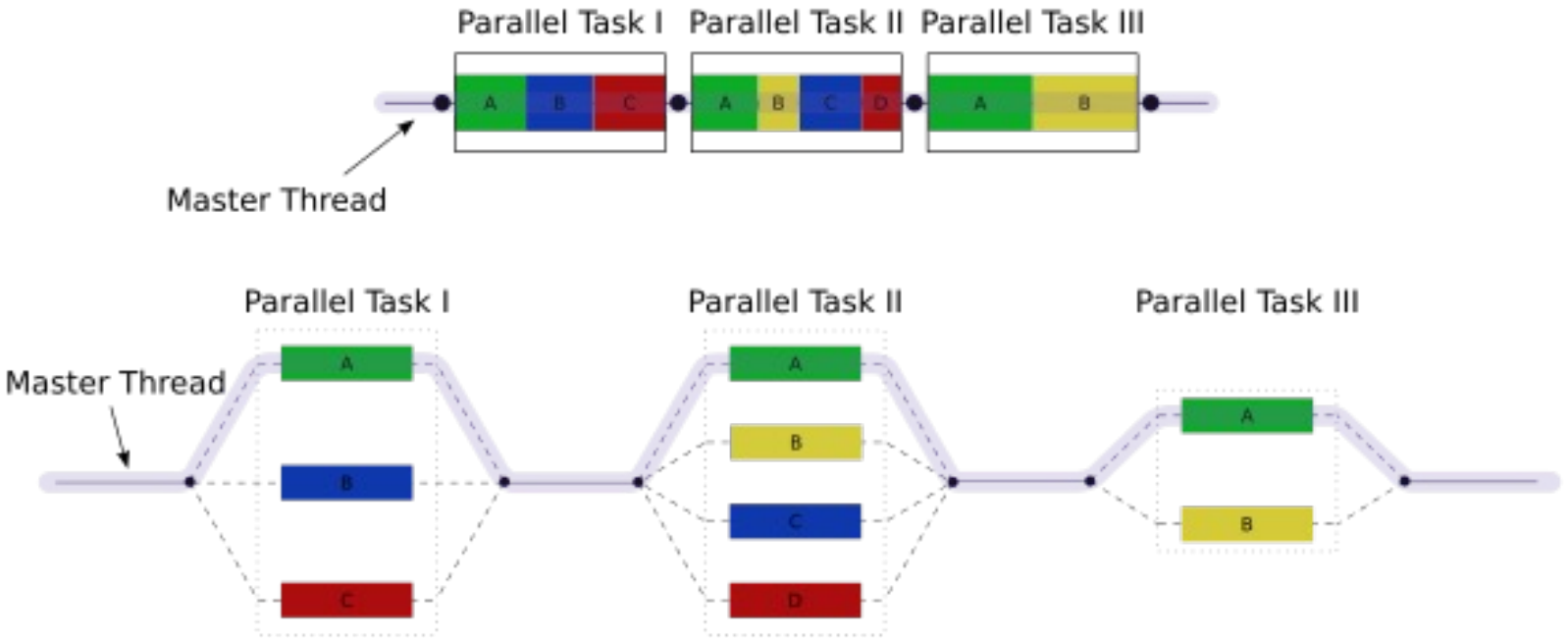
Concurrencia

- Se ejecutan tareas de manera simultanea, entrelazada, *interleaving*.
- No se requieren múltiples procesadores y es por eso que los primeros sistemas operativos lo implementaron: Corrutinas.



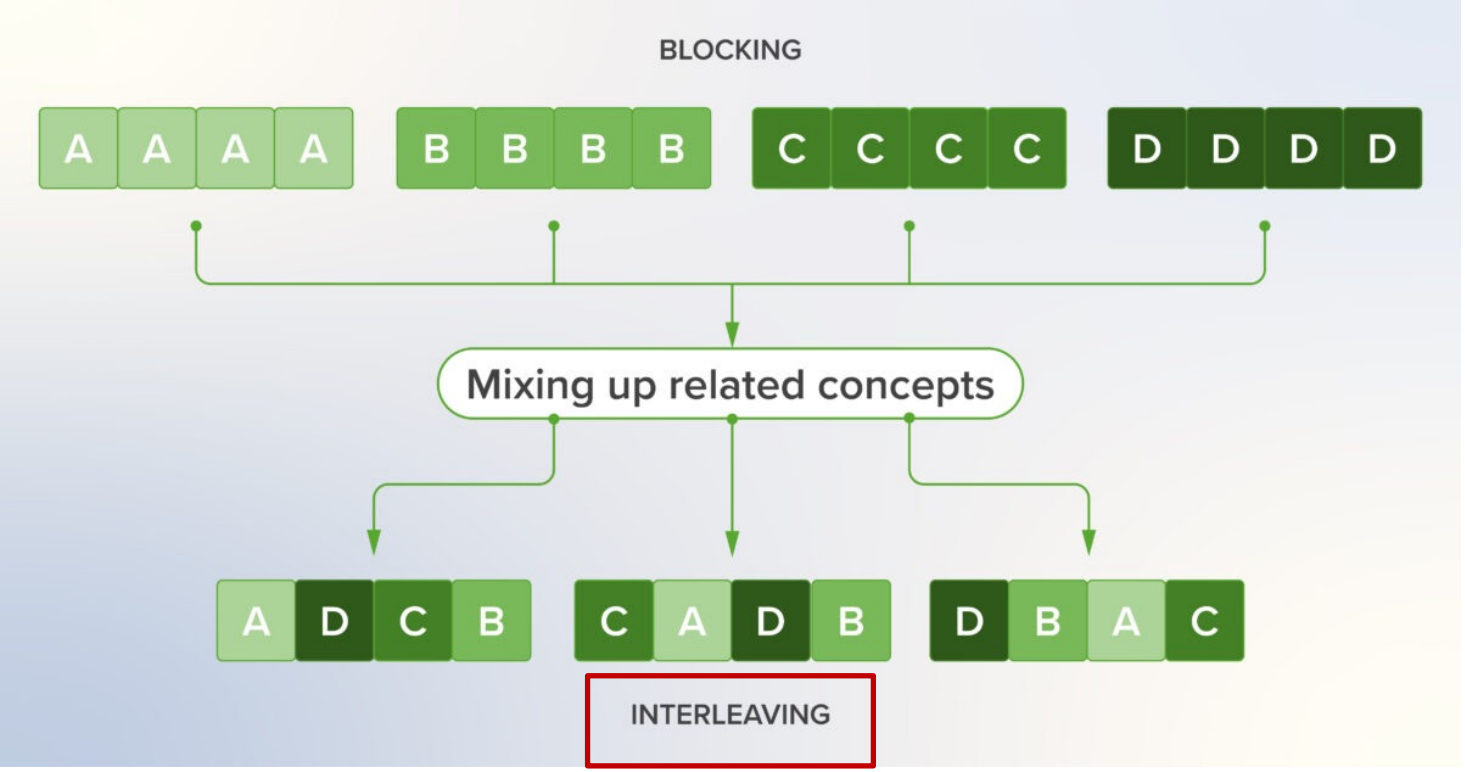
Entidades Concurrentes

Representaciones gráficas



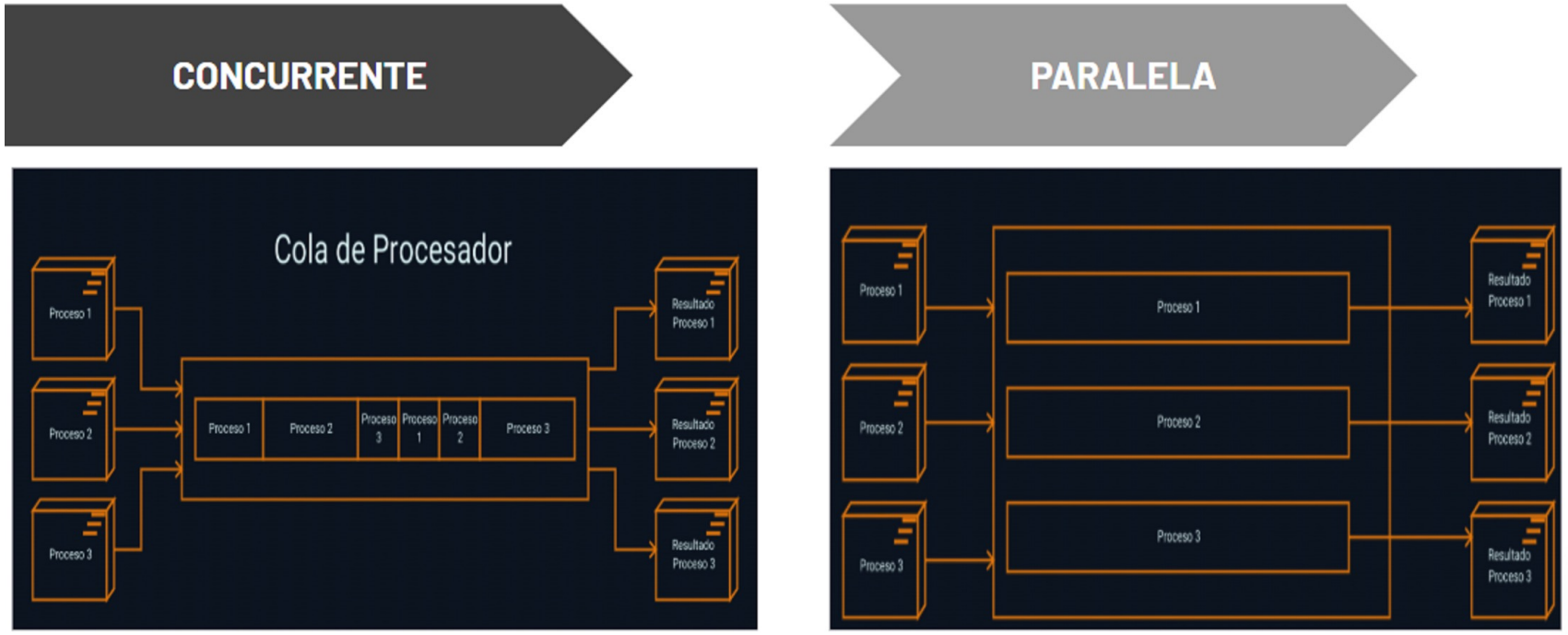
Entidades Concurrentes

Representaciones gráficas



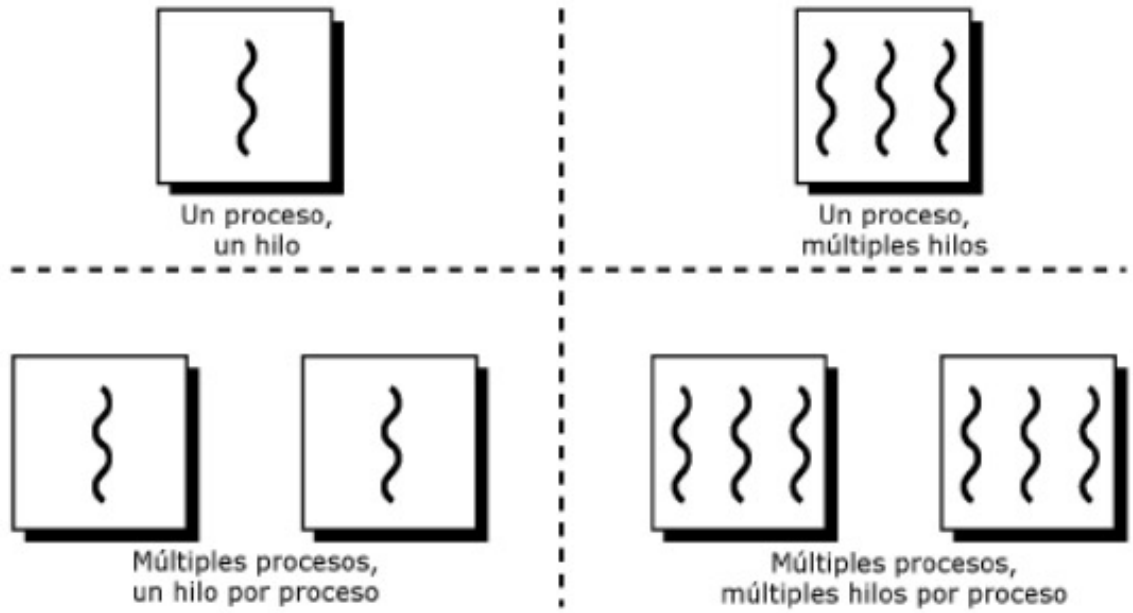
Entidades Concurrentes

Representaciones gráficas



Entidades Concurrentes

Representaciones gráficas



Corrutines

Corrutinas

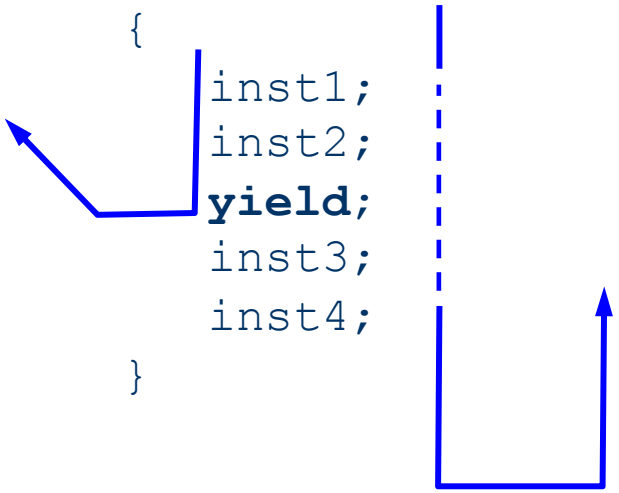
Corrutine

Corrutinas

Donald E. Knuth
Latex
Corrutinas
The Art of Programming

Una **corrutina** es como una función que tiene la capacidad de recomenzar su ejecución en puntos específicos de su código (con una instrucción especial) y no desde el inicio.

Ejemplo



En ocasiones `yield` puede regresar un valor.

La instrucción `yield` se puede ver como un cambio de contexto voluntario. En otras palabras se construye un ***Sistema Cooperativo***.

En UNIX existen las funciones `setjmp` y `longjump`
que sirven para implementar la noción de corrutina.

Tarea

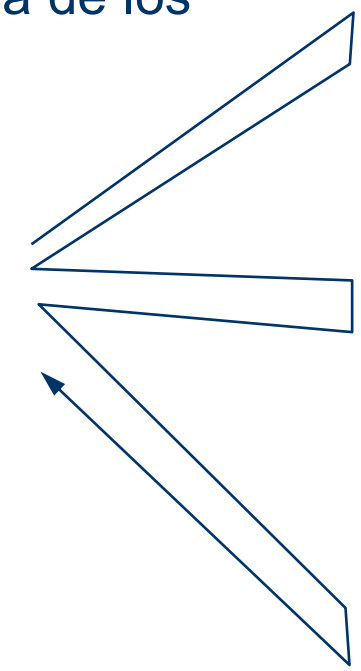
Ver la librería coro (http://www.goron.de/~froese/)
http://en.wikipedia.org/wiki/Coroutines

Corrutine

Programación

El objetivo es de construir un ***Scheduler Cooperativo*** y no un ***Scheduler Preemptivo*** como el que se utiliza con los procesos y la mayoría de los schedulers que utilizan Threads.

```
scheduler()
{
   while( Hay_tareas(cola) ){
     c = Elige_una_tarea(cola);
     v = c(); // Se ejecuta una parte
     if(v == FIN) Elim(cola, c);
     else Agrega(cola, c)
   }
}
```



```
cor1(param)
{
   inst1;
   yield val;
   inst2;
}
```

```
cor2(param)
{
   inst1;
   yield val;
   inst2;
}
```

```
corN(param)
{
   inst1;
   yield val;
   inst2;
}
```

Corrutine

Programación

Tarea optativa

- Hacer un programa que implemente corrutinas (N) y un scheduler.
- Implementar hasta donde sea posible el yield.

```
corrutina(parametros)
{
    inst1;
    inst2;
    yield;
    inst3;
    inst4;
}
```

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652