Manejo de Errores: *Programación Concurrente*

Luis Garreta
luis.garreta@javerianacali.edu.co

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

25 de octubre de 2017

PROGRAMACIÓN CONCURRENTE

Introducción

- El concepto fundamental de la programación concurrente es la noción de Proceso.
- Proceso: Cálculo secuencial con su propio flujo de control.
- La concurrencia en software implica la existencia de diversos flujos de control en un mismo programa colaborando para resolver un problema.

Procesos vs. Hilos

- En el contexto del Sistema Operativo, un Proceso es una instancia de un Programa que está siendo ejecutado en el ordenador.

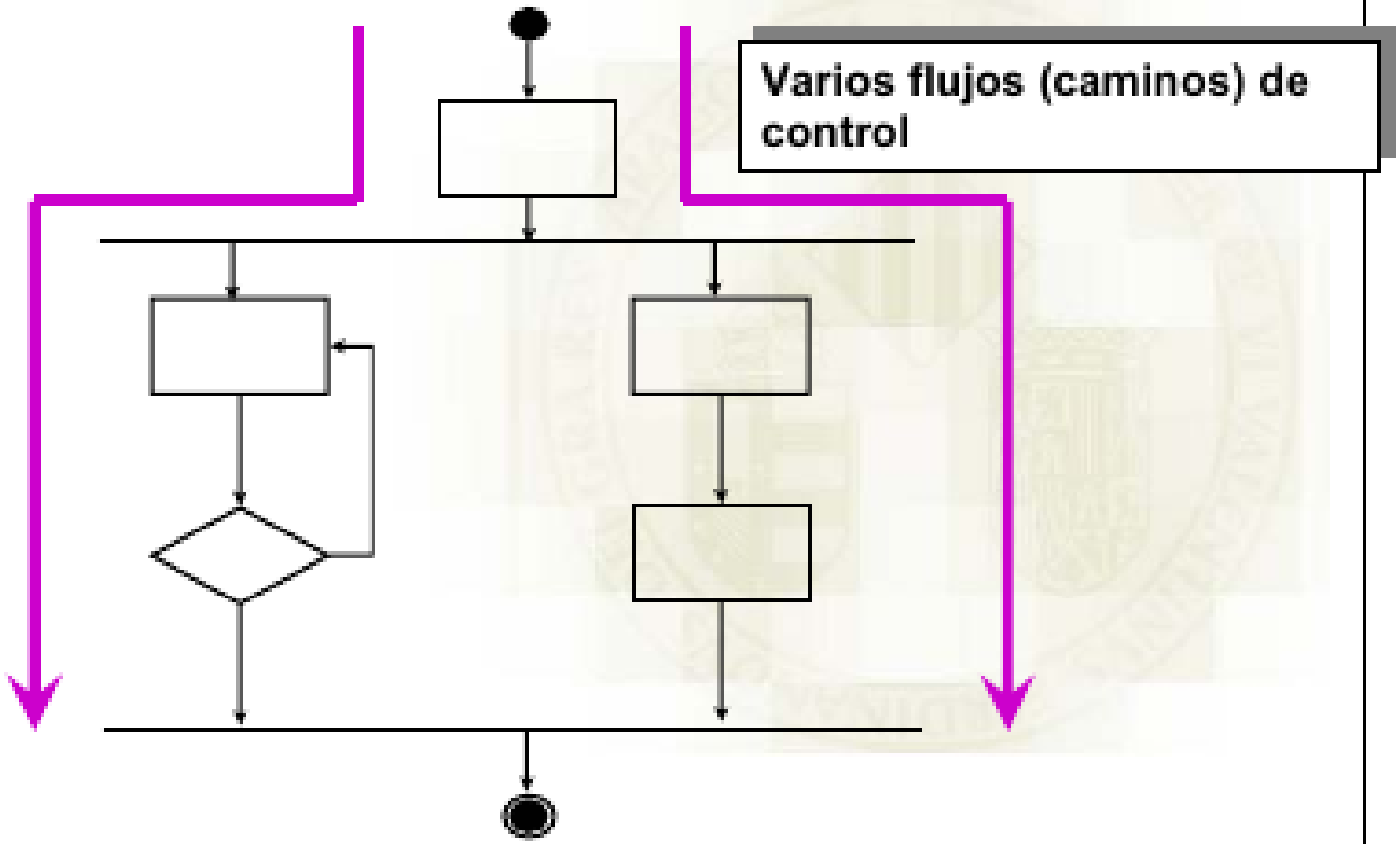
  Proceso = Código de programa + Datos + Recursos

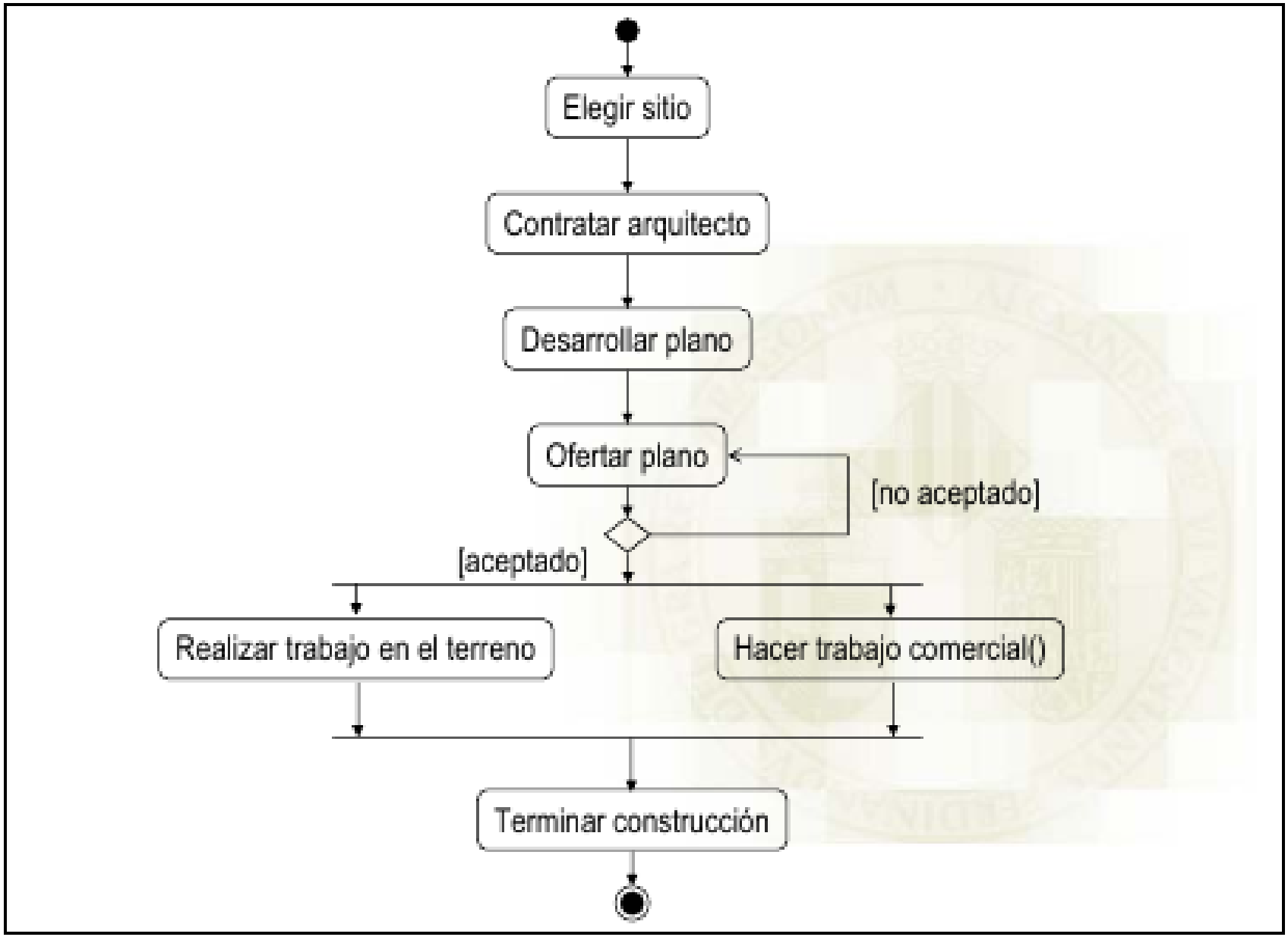
- Un S.O. admite concurrencia si es capaz de manejar diversos procesos simultáneamente.

- En el contexto de un Programa concurrente, un Hilo (*Thread*) es cada uno de los flujos secuenciales de control independientes especificados en el programa.

Procesos Concurrentes

- Un programa concurrente da lugar, durante su ejecución, a un proceso con varios hilos de ejecución.




Elegir sitio
Contratar arquitecto
Desarrollar plano
Ofertar plano
[no aceptado]
[aceptado]
Realizar trabajo en el terreno
Hacer trabajo comercial()
Terminar construcción

Concurrencia Software vs. Paralelismo Hardware

- **La concurrencia software es un concepto lógico, no implica la existencia de paralelismo en el hardware:**

  - ✓ Las operaciones hardware ocurren en paralelo si ocurren al mismo tiempo.

  - ✓ Las operaciones (software) en un programa son concurrentes si pueden ejecutarse en paralelo, aunque no necesariamente deben ejecutarse así.

Tipos de concurrencia

- **Concurrencia Física:**
  - ✓ Existe más de un procesador y varias unidades (hilos) de un mismo programa se ejecutan realmente de forma simultánea.

- **Concurrencia Lógica:**
  - ✓ Asumir la existencia de varios procesadores, aunque no existan físicamente. El implementador de tareas del lenguaje se encargará de "mapear" la concurrencia lógica sobre el hardware realmente disponible.

- **La concurrencia lógica es más general, pues el diseño del programa no está condicionado por los recursos de computación disponibles.**

Concurrencia en un Programa

- Desde el punto de vista de la programación, interesa la concurrencia lógica que puede existir en el interior de un programa.

- Un Lenguaje de Programación será concurrente si posee las estructuras necesarias para definir y manejar diferentes tareas (hilos de ejecución) dentro de un programa.
  - ✓ Ejemplos: Java, Ada

- El compilador y el SO serán los responsables de "mapear" la concurrencia lógica del programa sobre el hardware disponible.

C++ Multithreading

- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a thread,
- And each thread defines a separate path of execution.
- C++ does not contain any built-in support for multithreaded applications.
  - Instead, it relies entirely upon the operating system to provide this feature.
- The next example assumes that you are working on Linux OS
  - We are going to write multi-threaded C++ program using POSIX.
  - POSIX Threads, or Pthreads provides API which are available on many Unix-like POSIX systems such as:
    - FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris.

Creating Threads

- The following routine is used to create a POSIX thread:

```
1 #include <pthread.h>
2 pthread_create (thread, attr, start_routine, arg)
```

- Here, pthread_create creates a new thread and makes it executable.
- This routine can be called any number of times from anywhere within your code.
- Here is the description of the parameters:
  1. **thread:** An opaque, unique identifier for the new thread returned by the subroutine.
  2. **attr**: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  3. **start_routine**: The C++ routine that the thread will execute once it is created.
  4. **arg:** A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

Terminating Threads

- There is following routine which we use to terminate a POSIX thread –

```
1 #include <pthread.h>
2 pthread_exit (status)
```

- Here pthread_exit is used to explicitly exit a thread.
- Typically, the pthread_exit() routine is called after a thread has completed its work and is no longer required to exist.
- If main() finishes before the threads it has created, and exits with pthread_exit(), the other threads will continue to execute.
- Otherwise, they will be automatically terminated when main() finishes.

Ejemplo Threads en C++

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <pthread.h>
4  using namespace std;
5  #define NUM_THREADS 5
6  void *PrintHello(void *threadid) {
7     long tid = (long)threadid;
8     cout << "Hello World! Thread ID, " << tid << endl;
9     pthread_exit(NULL);
10 }
11 int main () {
12    pthread_t threads[NUM_THREADS];
13    int i,rc;
14
15    for( i = 0; i < NUM_THREADS; i++ ) {
16     cout << "main() : creating thread, " << i << endl;
17     rc = pthread_create(&threads[i],NULL,PrintHello,(void *)i);
18     if (rc) {
19      cout << "Error:unable to create thread," << rc << endl;
20      exit(-1);
21     }
22    }
23    pthread_exit(NULL);
24 }
```