Threads

Lenguajes de Programación I

Objetivos

- Evolución de la **Pila de Registros de Activación** en **Lenguajes tipo ALGOL** con **unidades concurrentes**
- Conceptos a recordar de prácticas previas
  - Lenguajes Tipo ALGOL
  - Pila de Registros de Activación
- Conceptos a introducir
  - Unidades concurrentes
  - Unidades concurrentes (en conjunto con los conceptos previos)

Unidades Concurrentes - Threads

- Semi-Proceso
  - Tiene su propia pila
  - Ejecuta una porción de código dada
- Pero
  - Todos los threads se encuentran incluídos dentro del mismo proceso
  - Comparten memoria entre sí.
  - Pueden acceder a las mismas variables globales
  - Heap compartido
  - Utilizan los mismos descriptores de Archivo

Al igual que los procesos

Threads - Ventajas

- Se ejecutan en Paralelo
  - Usando porciones de tiempo asignadas al proceso general
- Muchas operaciones se pueden llevar a cabo en forma paralela.
- Los eventos asociados a cada actividad se pueden atender tan pronto como sea necesario.
- Cambio de contexto mucho mas rápido que en el caso de procesos
  - descriptores, registros y demás permanecen sin cambios entre un cambio de contexto de threads
- las comunicaciones entre dos threads son usualmente más rápidas y sencillas de implementar que las comunicaciones entre dos procesos.

Threads - Ventajas

- Menos tiempo invertido en la creación y terminación de un thread que un proceso
- Menos tiempo en el cambio de contexto
- Al compartir el espacio de direcciones, la información es muy simple de compartir entre threads

Threads - Desventajas

- Justamente por su ventaja y debido a que comparten el mismo espacio de direcciones, el comportamiento se puede volver inestable.
- En el caso de procesos, el SO protege el espacio de direcciones del acceso de otros procesos.

Threads - Implementaciones

- A Nivel Lenguaje (Nivel Usuario)
  - El kernel del sistema operativo no se preocupa por la existencia de los threads
  - La administración de los threads es realizada por la aplicación, usando alguna biblioteca de threads.
- A Nivel Sistema Operativo
  - La administración es realizada por el kernel del SO
  - Información de contexto de los threads y procesos mantenidos por el kernel

Threads – Nivel Lenguaje

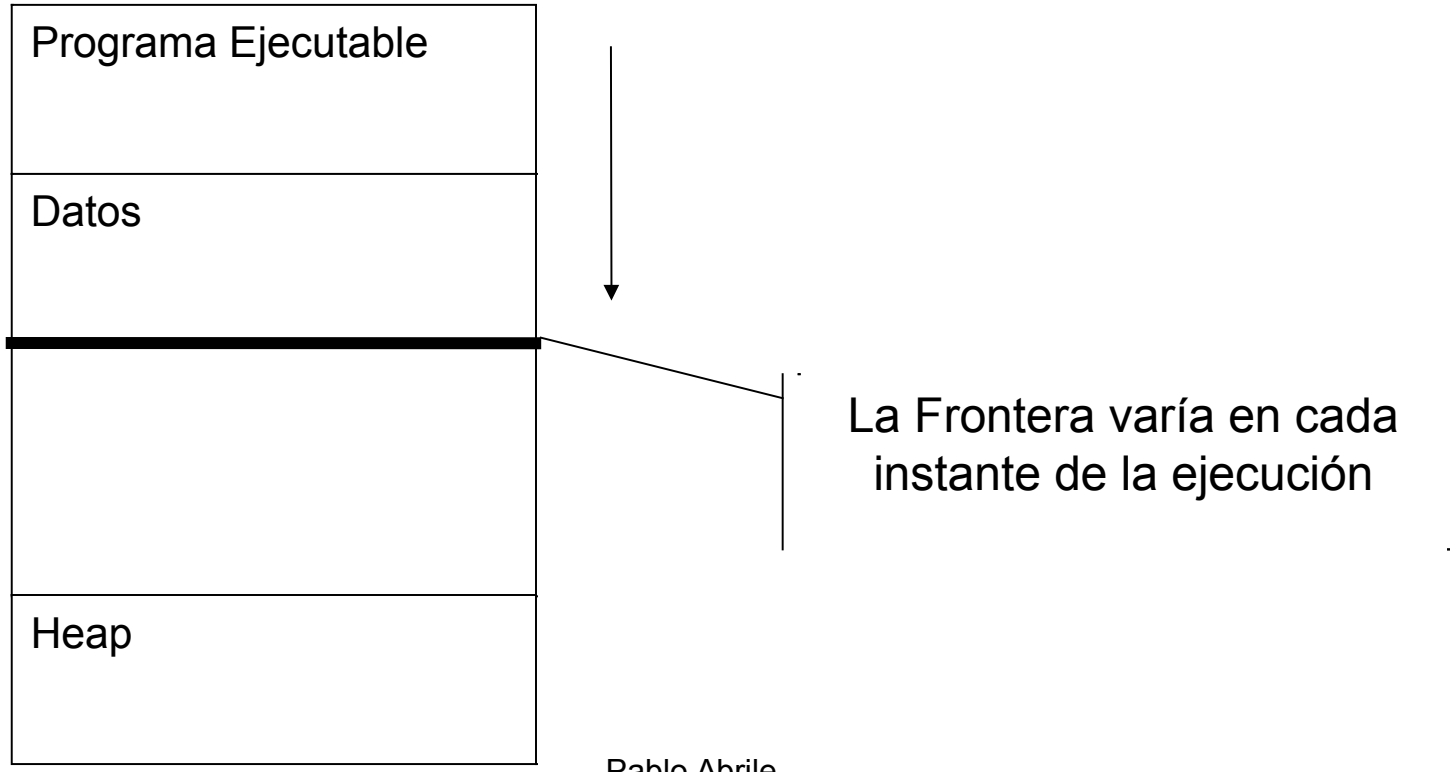
- Ventajas
  - El cambio entre threads no involucra a sistema operativo.
  - La planificación puede ser especifica de la aplicación
  - Pueden correr en cualquier sistema operativo
- Desventajas
  - El Kernel bloquea procesos
  - Sólo pueden asignarse procesadores a procesos: Dos threads dentro del mismo proceso no podrán ejecutarse en forma simultánea.

Threads – Nivel SO

- Ventajas
  - El kernel puede simultáneamente administrar varios threads de un mismo proceso en varios procesadores.
  - Las rutinas del kernel pueden ser multithreaded.
- Desventajas:
  - Los cambios entre threads dentro de un mismo proceso involucran la intervención del kernel → demoras en el tiempo de ejecución.

Lenguajes Tipo ALGOL

- También llamados Orientados a Pila



Lenguajes Tipo ALGOL

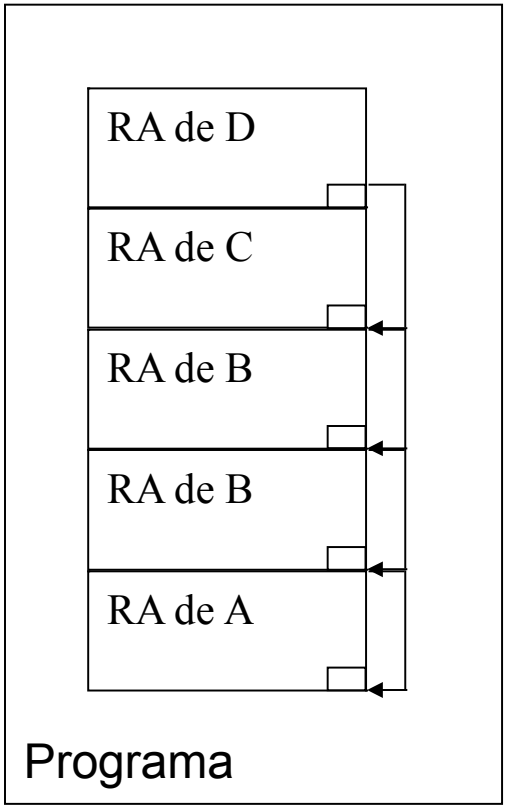
- Tienen bloques (fragmentos de programa)
  - Sin nombre
  - Con nombre (funciones)
- Dentro de estos bloques pueden definirse variables a emplear
- Al llamar a estos bloques, en la pila de ejecución se crean los registros de activación donde se almacenan las variables que se emplean

Pila de Registros de Activación

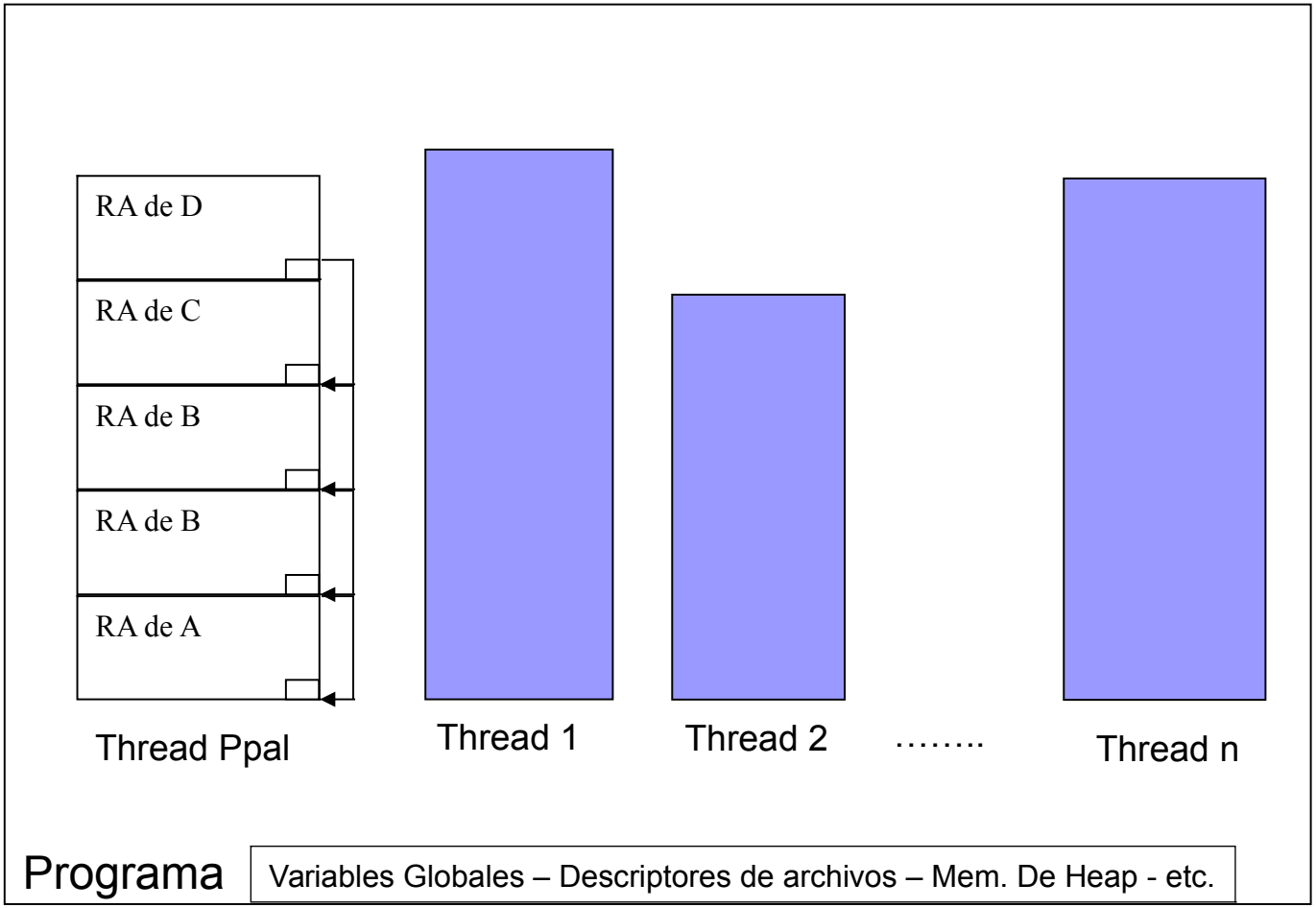
- Registro de Activación
  - Conjunto de todos los elementos nuevos que necesita un bloque para ejecutarse.
    - Variables
    - Dirección de retorno si llamo a una función (cadena dinámica)
    - Puntero al padre en el árbol de anidamiento (cadena estática)

Pila de Registros de Activación

- A → B → B → C → D



Pila de Registros de Activación Unidades Concurrentes



Ejercicio Ejemplo

- Programa Ejecutable

```
#include <pthread.h>
#include <stdio.h>

void *XX(void) {
        int i;
        printf("%x\n",&i);
}
   int main (){
        pthread_t thread;
        pthread_create(&thread, NULL, XX, NULL);
        XX();
        XX();
        XX();
   }
```

Headers de la librería POSIX Threads

Función XX
Será llamada desde
-Thread creado con pthread_create
- Thread Principal (main)

Thread Principal

Creación del Thread que llamará a XX

Invocaciones a XX desde el thread principal

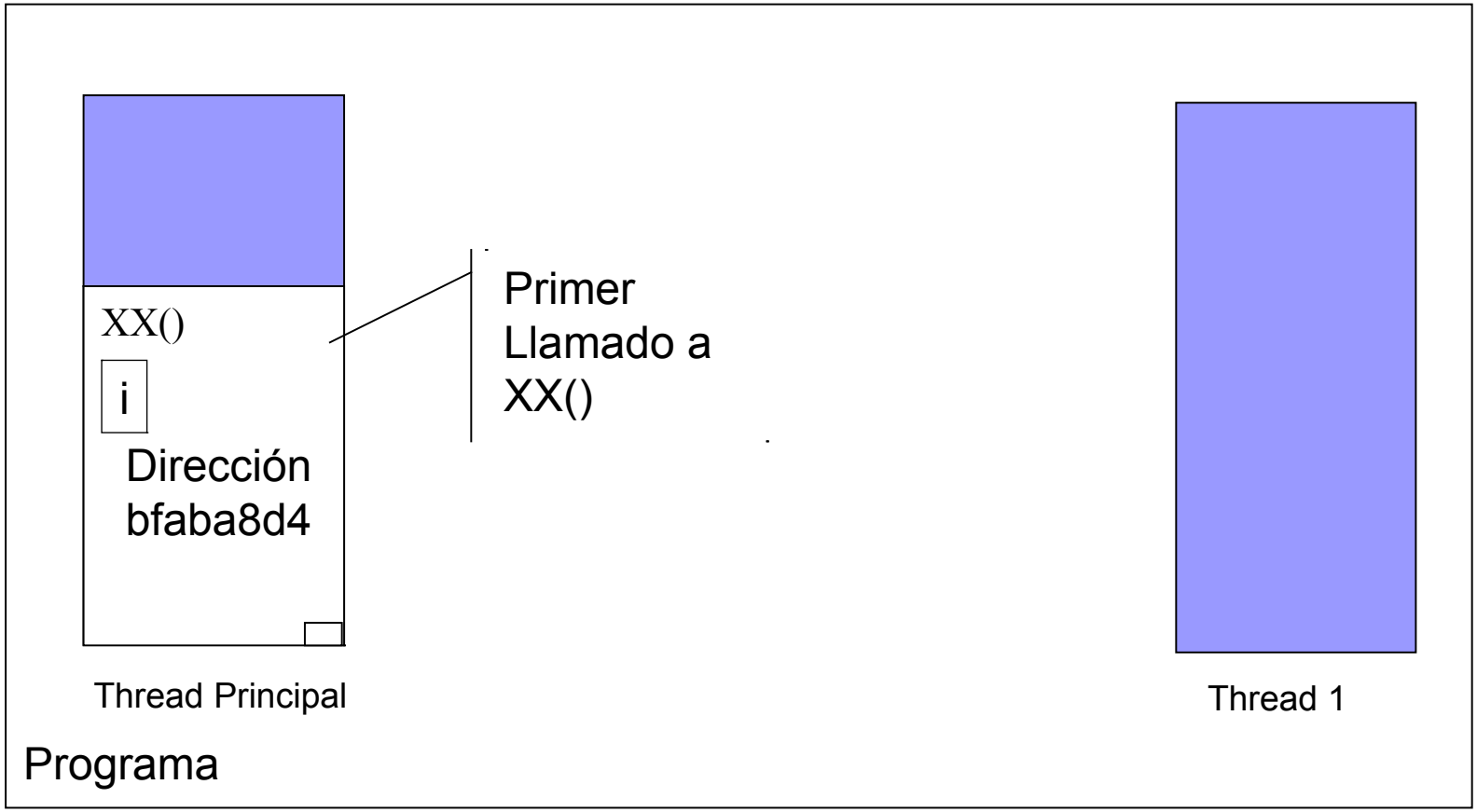
Salida Esperada

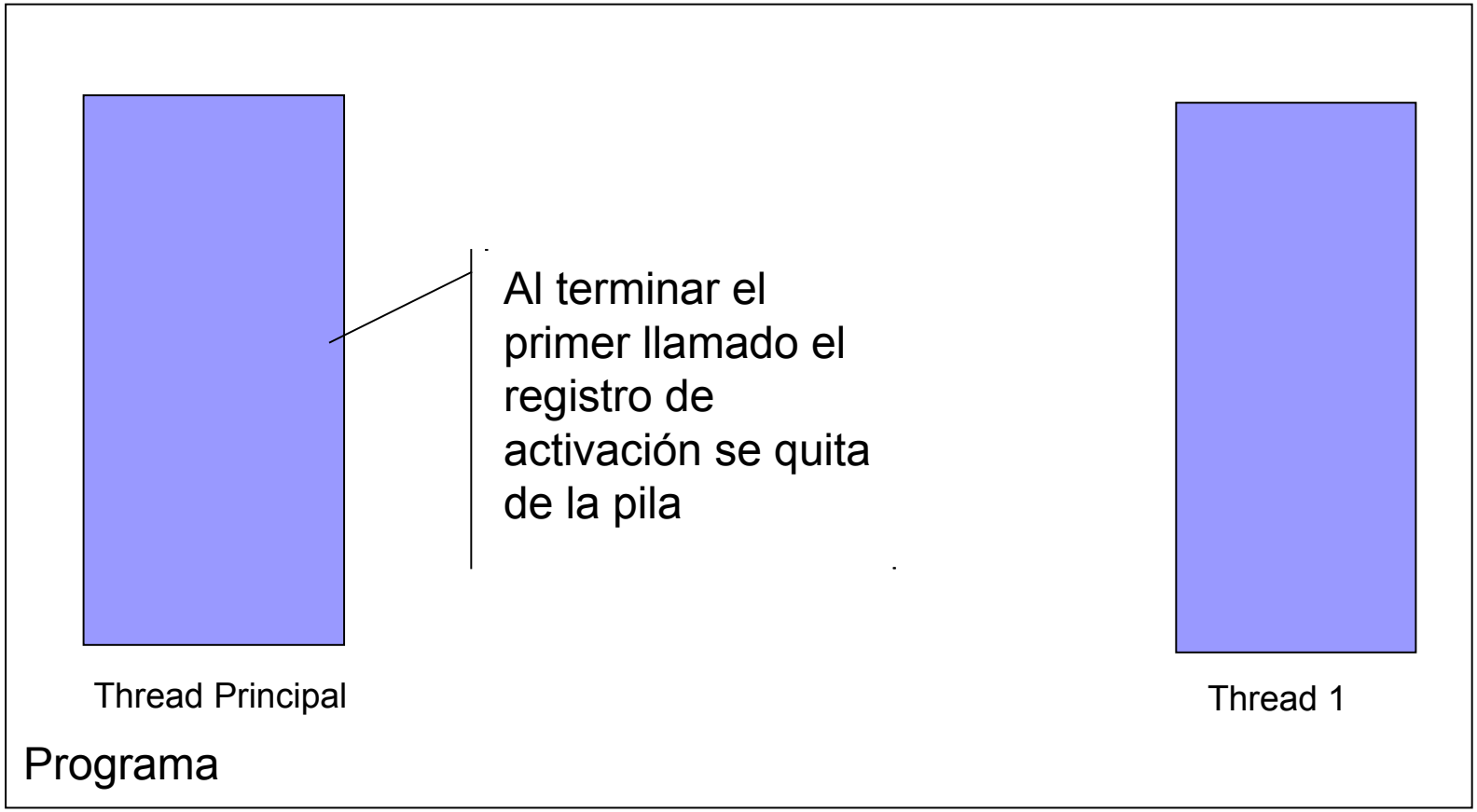
- XX se llama 4 veces en total
  - 3 desde el main. En adelante *Thread Principal*
  - 1 desde el thread creado con **pthread_create.** En adelante *Thread 1*
  - Se esperan: 4 impresiones por pantalla
- ¿Que ocurre con las direcciones de memoria que serán impresas?

Pilas de Ejecución

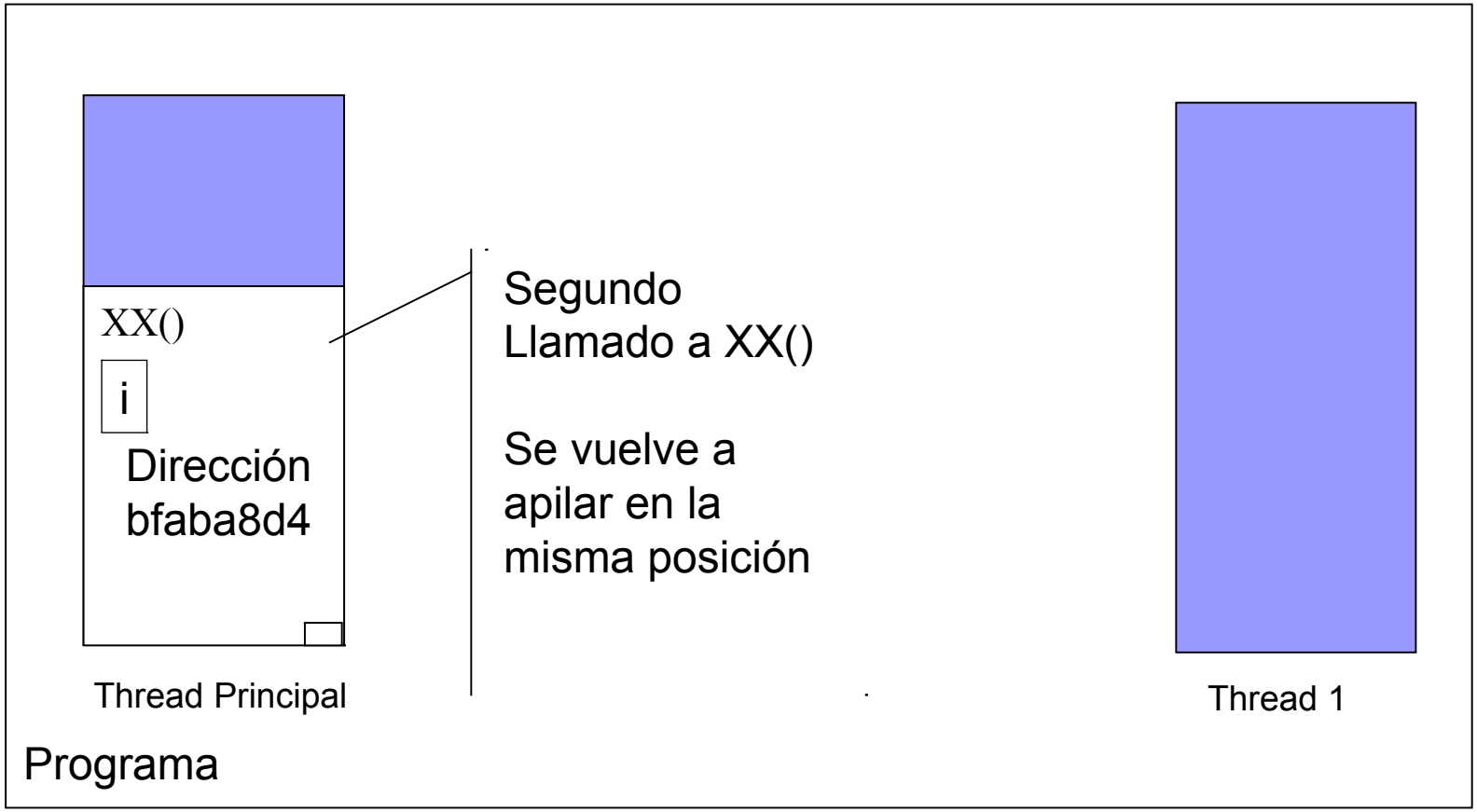
- 1 para el Thread Principal (main)
- 1 para el Thread 1 (creado por pthread_create)

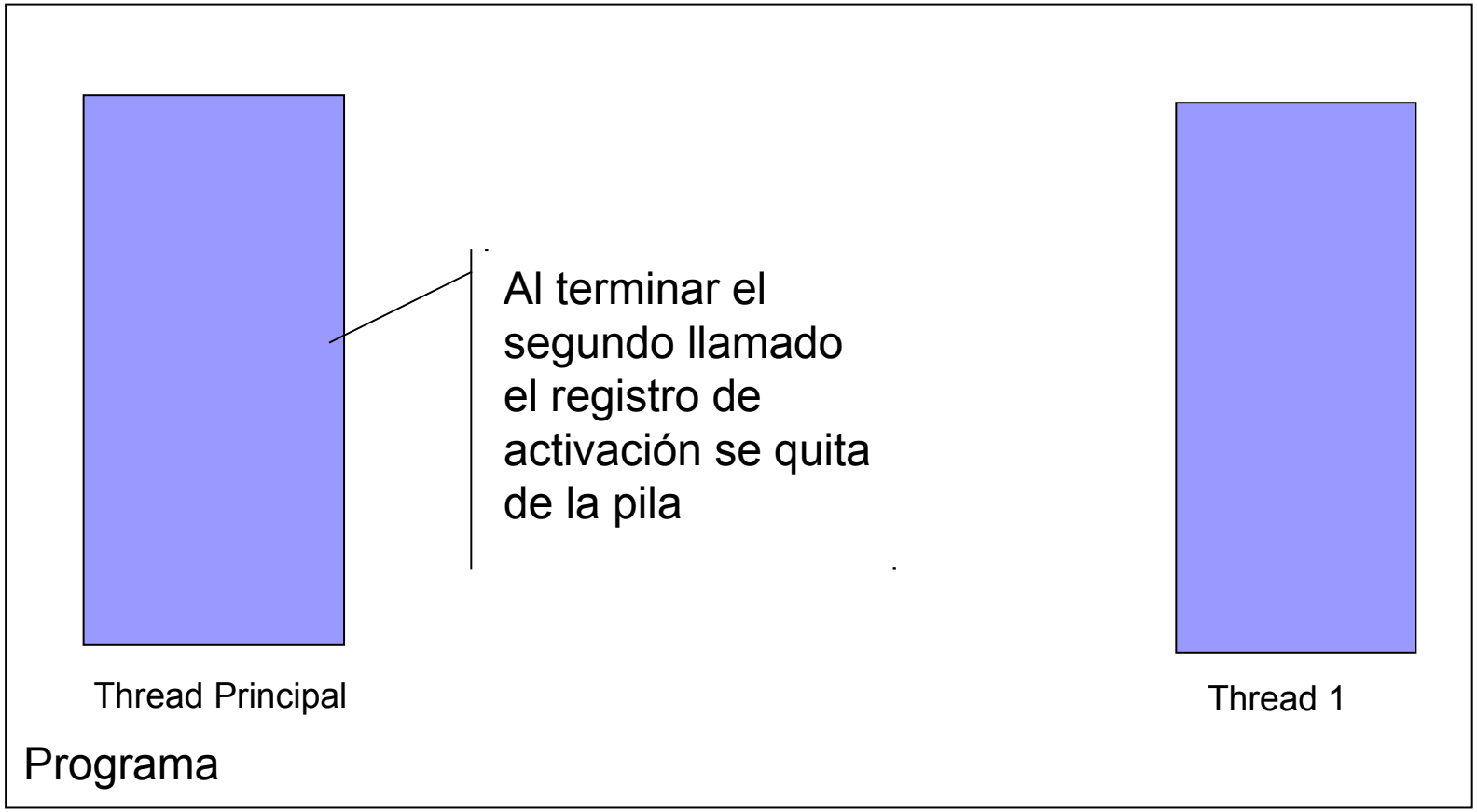
XX() (thread principal)



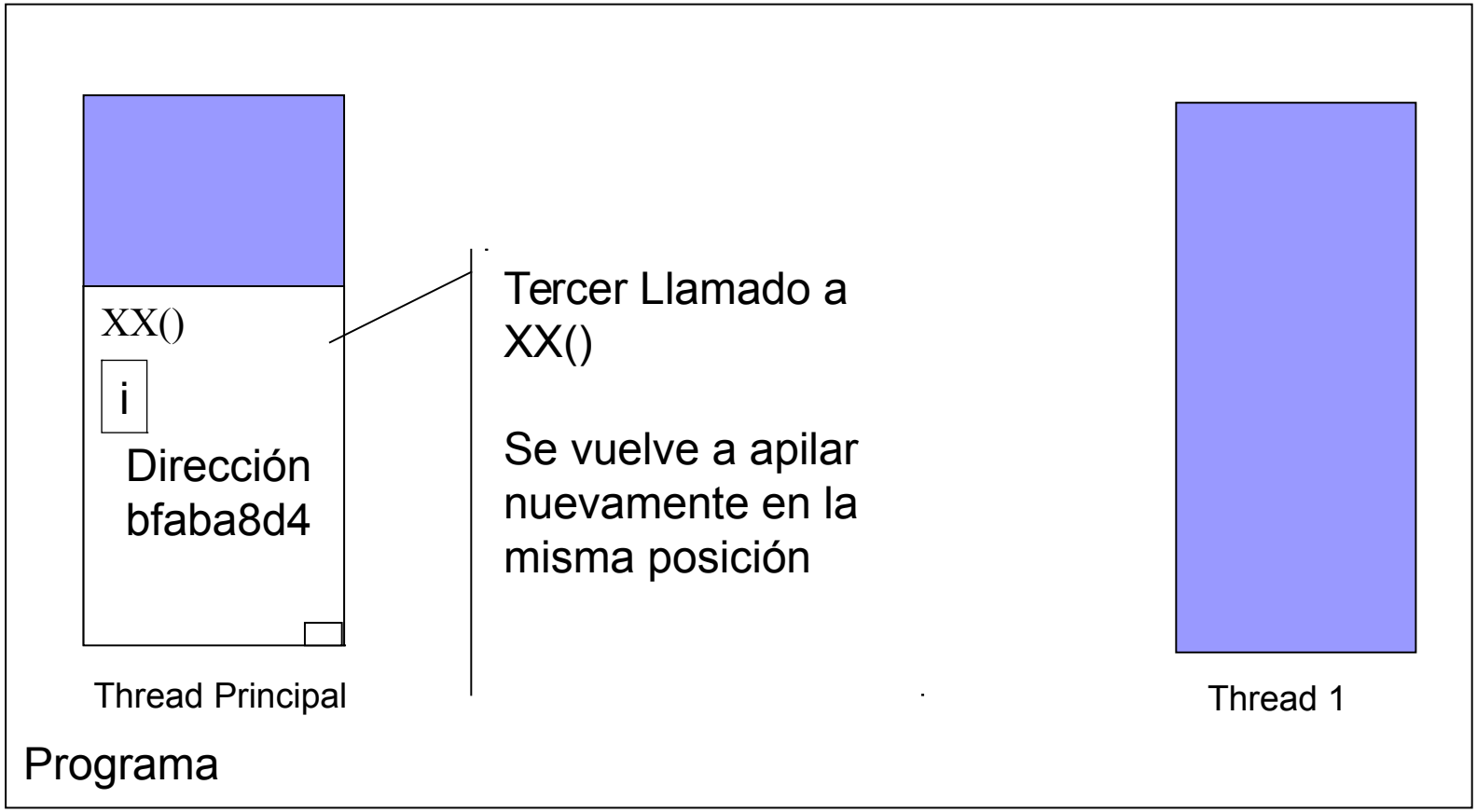

Al terminar el primer llamado el registro de activación se quita de la pila
Thread Principal
Thread 1
Programa

XX() (thread principal)

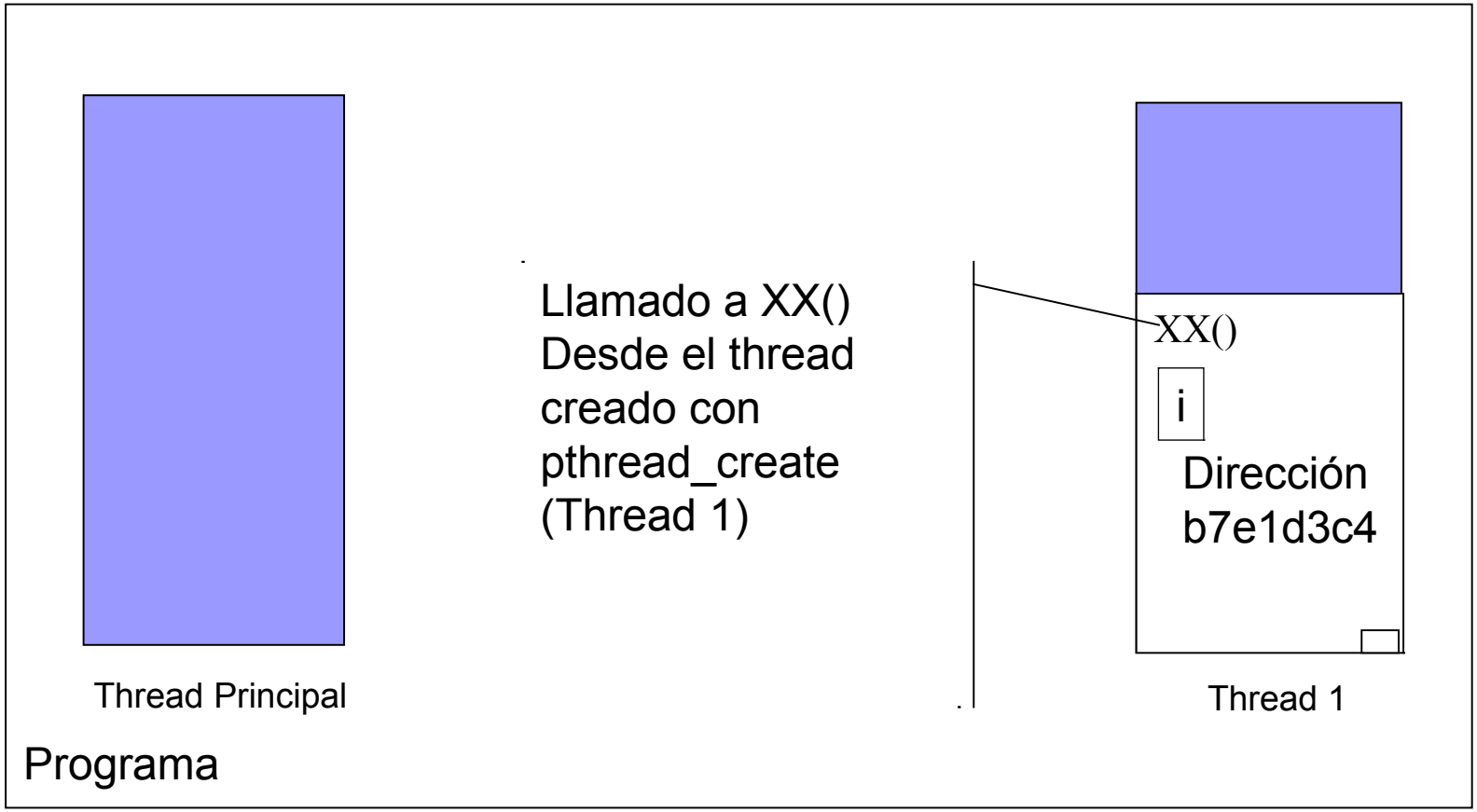



Al terminar el
segundo llamado
el registro de
activación se quita
de la pila
Thread Principal
Thread 1
Programa

XX() (thread principal)



XX() Thread 1

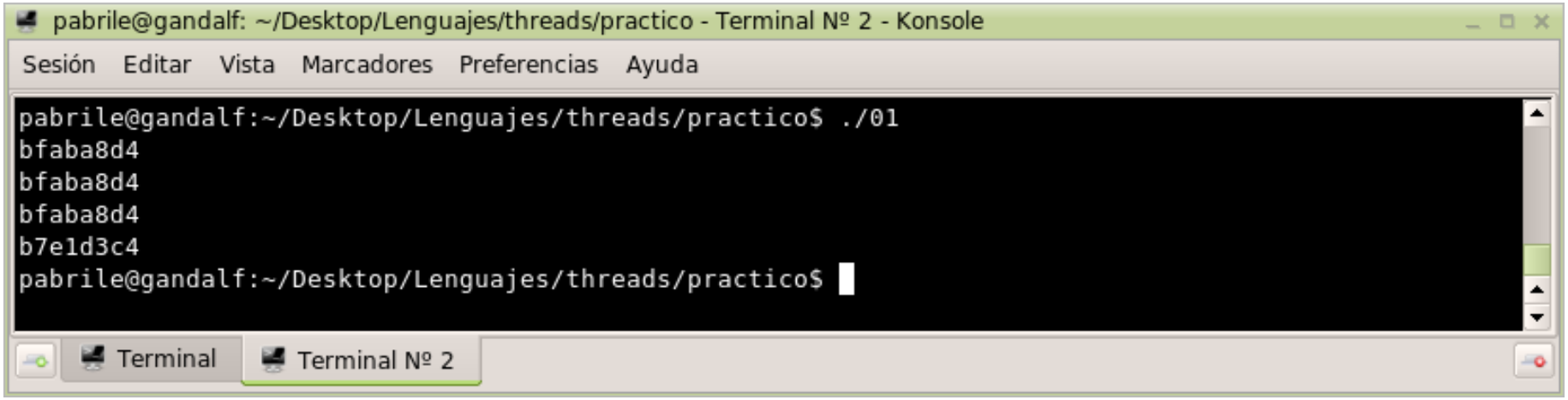

Llamado a XX()
Desde el thread
creado con
pthread_create
(Thread 1)
XX()
i
Dirección
b7e1d3c4
Thread Principal
Thread 1
Programa

Salida Esperada

- bfaba8d4
- bfaba8d4
- bfaba8d4
- b7e1d3c4

- El orden puede variar de acuerdo a cuando se ejecute el Thread 1
- Puede hacerlo entre las llamadas a XX() desde el Thread Principal

Salida Obtenida

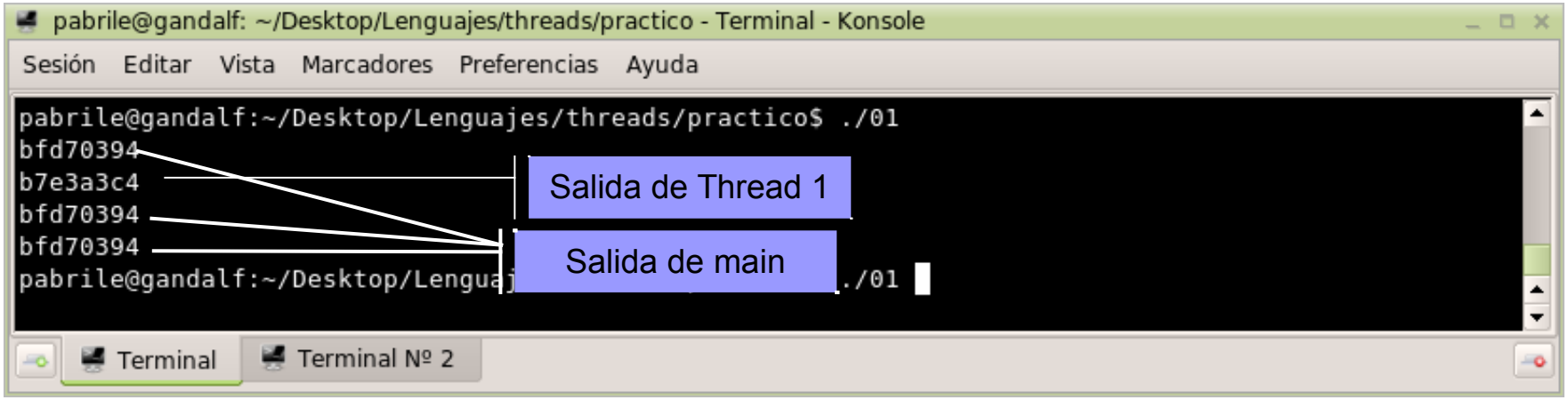


Orden de aparición alterno

- Introducimos una demora en la función XX
- Entre los tres llamados del thread principal a XX se alternará el llamado al thread creado con pthread_create

```
void *XX(void){
        int i;
        for (i=0; i<999999999; i++); //Demora antes de imprimir.
        printf("%x\n",&i);
}
```

Salida



- Debido a la demora, se alternan las salidas
- **El resultado sigue siendo el mismo ya que las pilas de ejecución son separadas para cada thread!!**

Ejercicio 2

- 2 threads
  - ThreadHola: Imprime Hola
  - ThreadMundo: Imprime Mundo

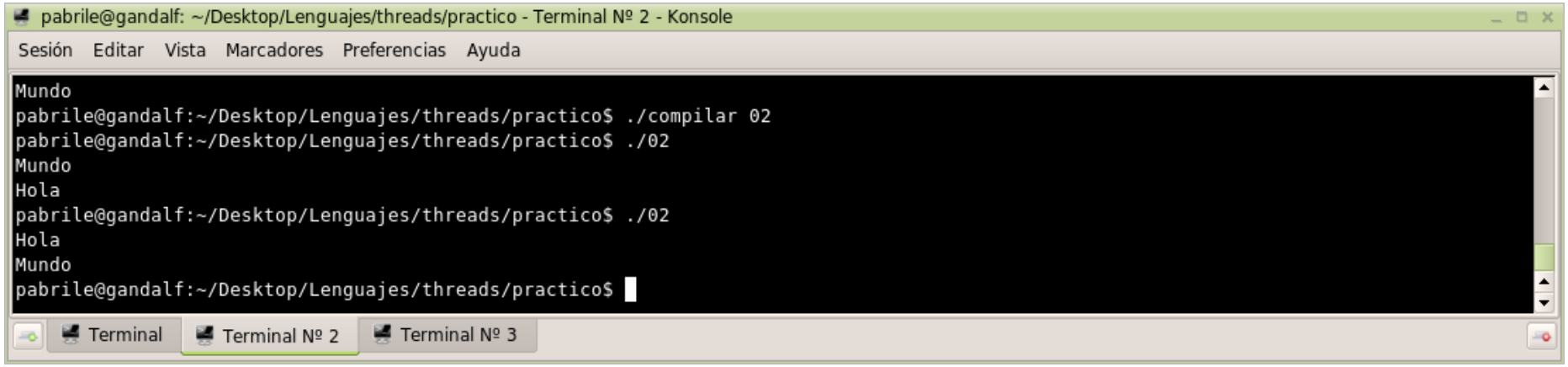
```
#include <pthread.h>
#include <stdio.h>

void *mensaje(void* msj){
        printf("%s\n",msj);
}
int main (){
        pthread_t threadHola,threadMundo;
        pthread_create(&threadHola, NULL, mensaje, "Hola");
        pthread_create(&threadMundo, NULL, mensaje, "Mundo");
}
```

El cuarto argumento de pthread_create es el argumento a pasar a **mensaje** cuando se crea el thread.
En este caso, el mensaje a imprimir por el thread

Resultado

- En general, por el poco tiempo de procesamiento que lleva la función “mensaje” se verá la respuesta “Hola” “Mundo” (aunque puede no ser así).
- Introducimos una demora en la función mensaje para ver si el orden de impresión puede variar

```
void *mensaje(void* msj){
        int i;
        for (i=0; i<999999999; i++);  //Demora introducida
        printf("%s\n",msj);
}
```



Un Ejercicio un poco más difícil

```
#include <pthread.h>
#include <stdio.h>
static __thread int y=5;
void *XX(void *threadid)
{
    int j,z;
    if (y<1)  return;
    else{
        for (j=0;j<99999999;j++) /*DEMORA*/;
        printf("%s Valor de y : %d\n",(char*)threadid,y);
        printf("%s distancia de y a z: %ld\n",(char*)threadid,&y-&z);
        y--;
        printf("%s posicion de Y : %p\n",(char*)threadid,&y);
        XX(threadid);
    }
}
```

Un Ejercicio un poco más difícil

```
int main ()
{
    int x;
    pthread_t thread1,thread2;
    pthread_create(&thread1, NULL, XX, (void*)"Thread1");
    pthread_create(&thread2, NULL, XX, (void*)"Thread2");
    pthread_exit(NULL);
}
```

```
Thread1 Valor de y: 5
Thread1 distancia de y a z: 533
Thread1 posicion de Y: 0x9f06fc
Thread2 Valor de y:5
Thread2 distancia de y a z : 533
Thread2 posicion de Y: 0x1ef6fc
Thread1 Valor de y: 4
Thread1 distancia de y a z : 545
Thread1 posicion de Y: 0x1ef6fb
Thread2 Valor de y: 4
Thread2 distancia de y a z : 545
Thread2 posicion de Y: 0x1ef6fc
Thread1 Valor de y: 3
Thread1 distancia de y a z : 557
Thread1 posicion de Y: 0x9f06fc
Thread2 Valor de y: 3
Thread2 distancia de y a z : 557
Thread2 posicion de Y: 0x1ef6fc
Thread1 Valor de y: 2
Thread1 distancia de y a z : 569
Thread1 posicion de Y: 0x1ef6fc
Thread2 Valor de y: 2
Thread2 distancia de y a z : 569
Thread2 posicion de Y: 0x1ef6fc
Thread2 Valor de y: 1
Thread2 distancia de y a z : 581
Thread2 posicion de Y: 0x1ef6fc
Thread1 Valor de y: 1
Thread1 distancia de y a z : 581
Thread1 posicion de Y: 0x9f06fb
```

Incorrecta

Motivo: La posición de Y no debe variar en la ejecución de todo el thread por su declaración como static __thread int.
En varias de las salidas se modifica esta dirección
(puede deducirse también por la distancia).

```
Thread1 Valor de y          :  5
Thread1 distancia de y a z:    533
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          : 5
Thread2 distancia de y a z  :  533
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  4
Thread1 distancia de y a z  :  525
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  4
Thread2 distancia de y a z  :  525
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  3
Thread1 distancia de y a z  :  517
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  3
Thread2 distancia de y a z :  517
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  2
Thread1 distancia de y a z :  509
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  2
Thread2 distancia de y a z :  509
Thread2 posicion de Y       :  0x1ef6fc
Thread2 Valor de y          :  1
Thread2 distancia de y a z :  491
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  1
Thread1 distancia de y a z :  491
Thread1 posicion de Y       :  0x9f06fc
```

Incorrecta

Motivo: Tamaño de registros de activación diferente en cada ejecución (por la distancia de las variables)
Si fuera correcta implicaría direcciones crecientes ya que Z se acercaría a Y en cada paso. Al ritmo de crecimiento no podrían ejecutarse mas de 100 iteraciones, lo cual no tiene sentido..

```
Thread1 Valor de y          :  5
Thread1 distancia de y a z:    533
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          : 5
Thread2 distancia de y a z  :  532
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  4
Thread1 distancia de y a z  :  545
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  4
Thread2 distancia de y a z  :  544
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  3
Thread1 distancia de y a z  :  557
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  3
Thread2 distancia de y a z :  556
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  2
Thread1 distancia de y a z :  569
Thread1 posicion de Y       :  0x9f06fc
Thread2 Valor de y          :  2
Thread2 distancia de y a z :  568
Thread2 posicion de Y       :  0x1ef6fc
Thread2 Valor de y          :  1
Thread2 distancia de y a z :  581
Thread2 posicion de Y       :  0x1ef6fc
Thread1 Valor de y          :  1
Thread1 distancia de y a z :  580
Thread1 posicion de Y       :  0x9f06fc
```

Incorrecta

Motivo: Las distancias entre z e y deben ser las mismas en cada iteración de cada thread ya que la estructura de los programas y el tamaño de los registros de activación son los mismos

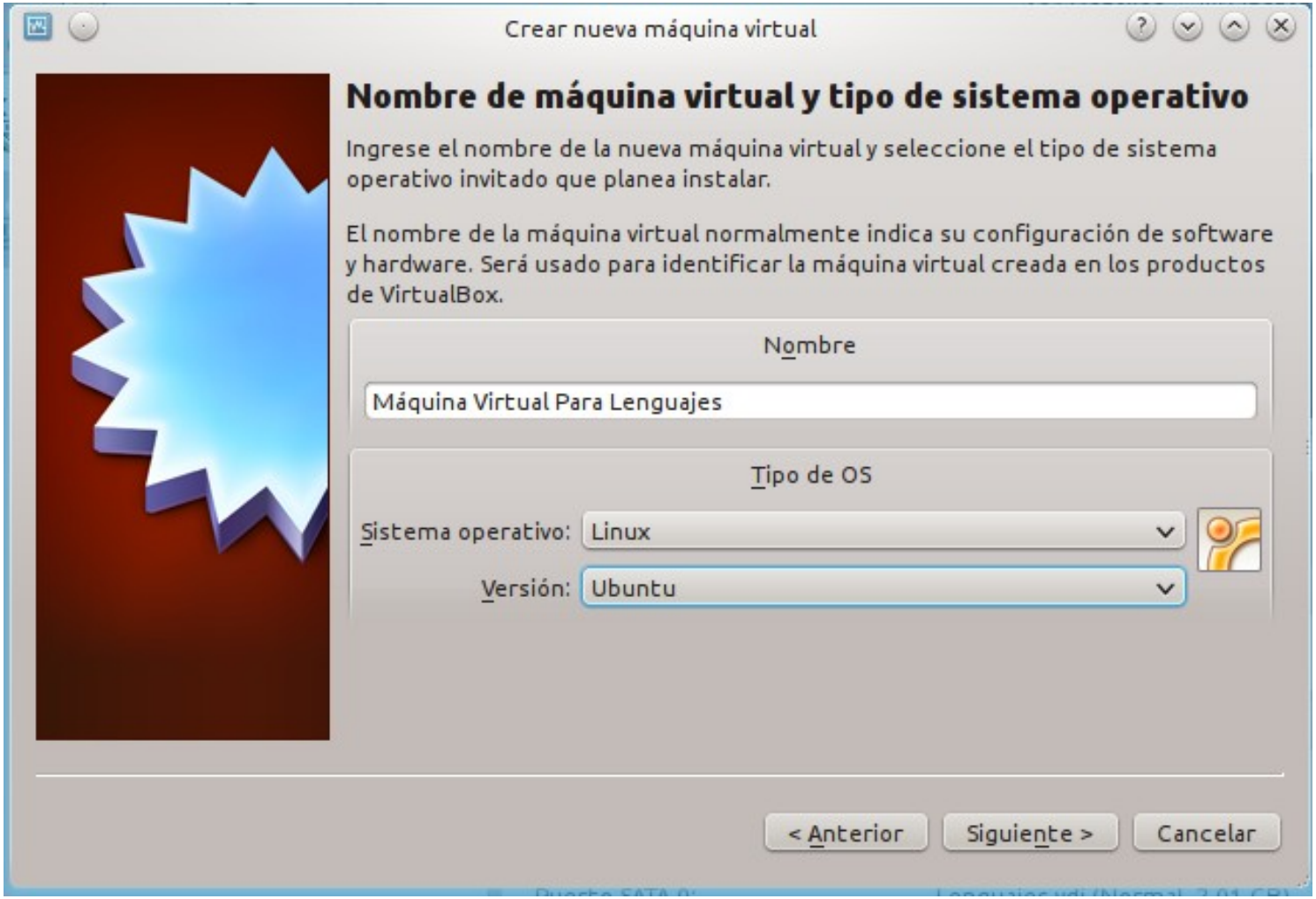
Para probar los ejercicios

- Comprobar que se tienen instaladas las librerías de POSIX threads.
  - En ubuntu: *sudo apt-get install libc6-dev gcc*
- Realizar un script “compilar” que contenga
  - gcc -w $1.c -o $1 –lpthread
- ./compilar 01_threads
  - Compila 01_threads.c
- ./01_threads

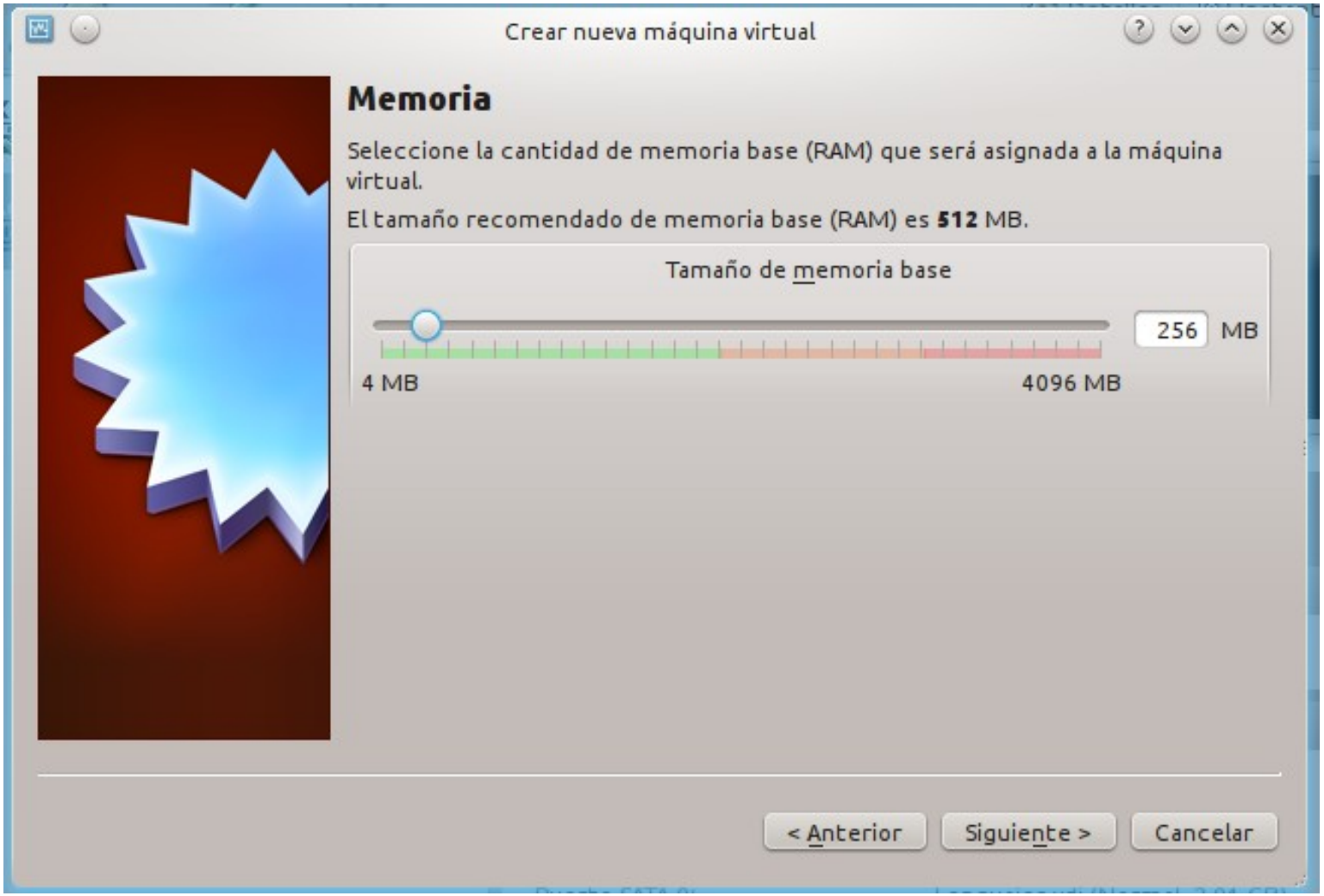
Para probar los ejercicios

- Emplear la máquina virtual provista por la cátedra en su página.
- Seguir los pasos de creación a partir de VirtualBox

Instalación con VirtualBox

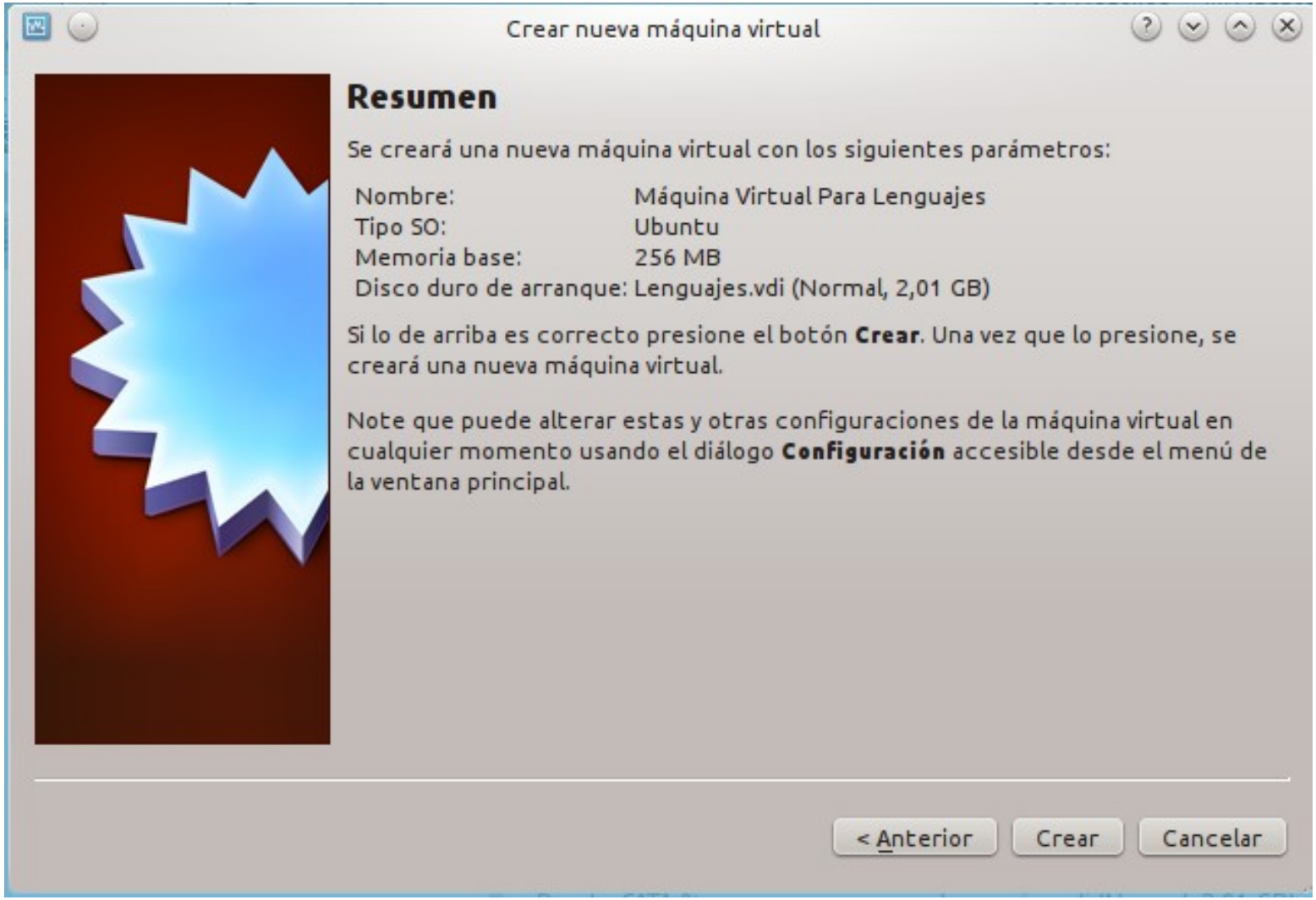


Instalación con VirtualBox

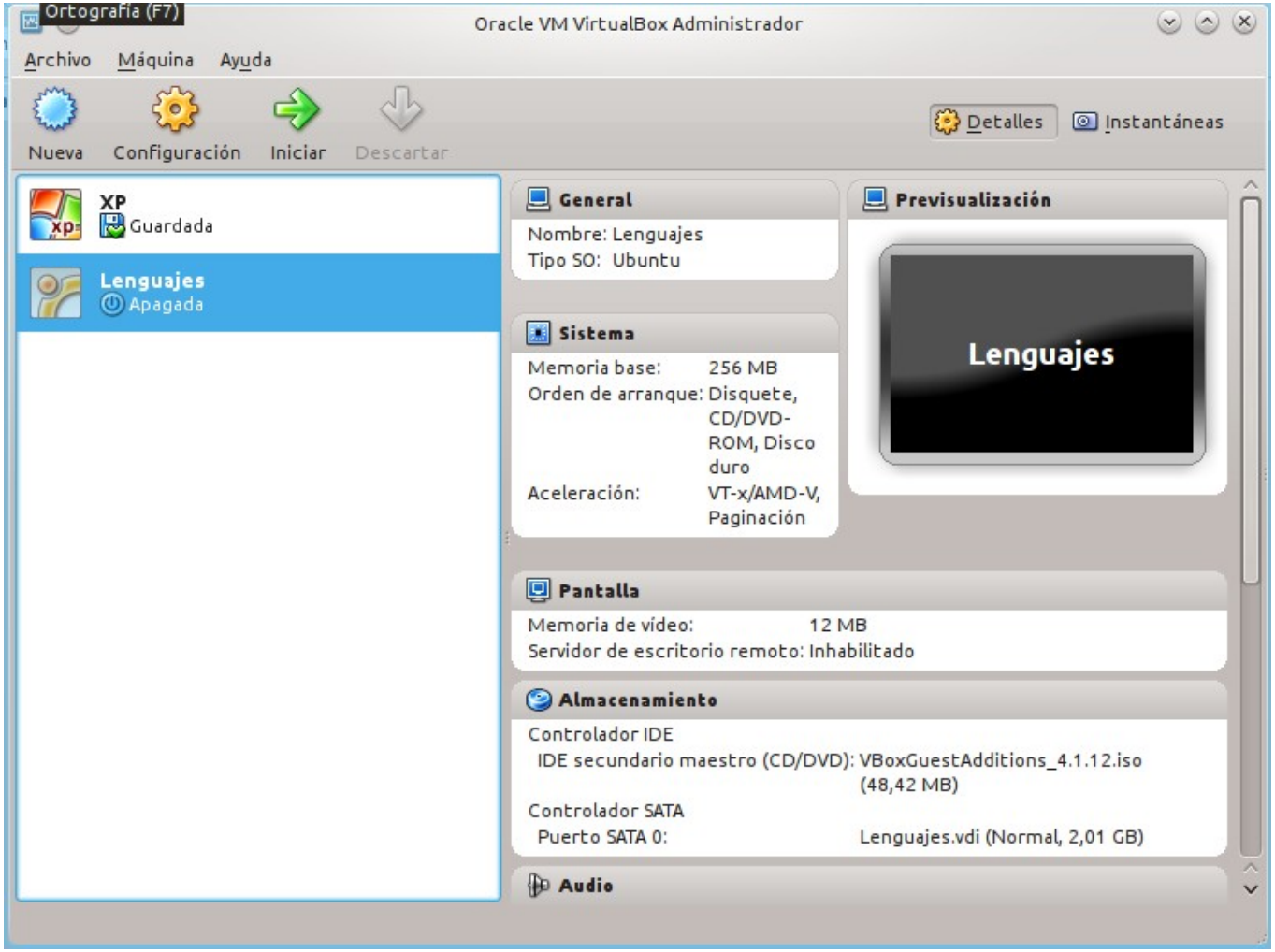


Instalación con VirtualBox

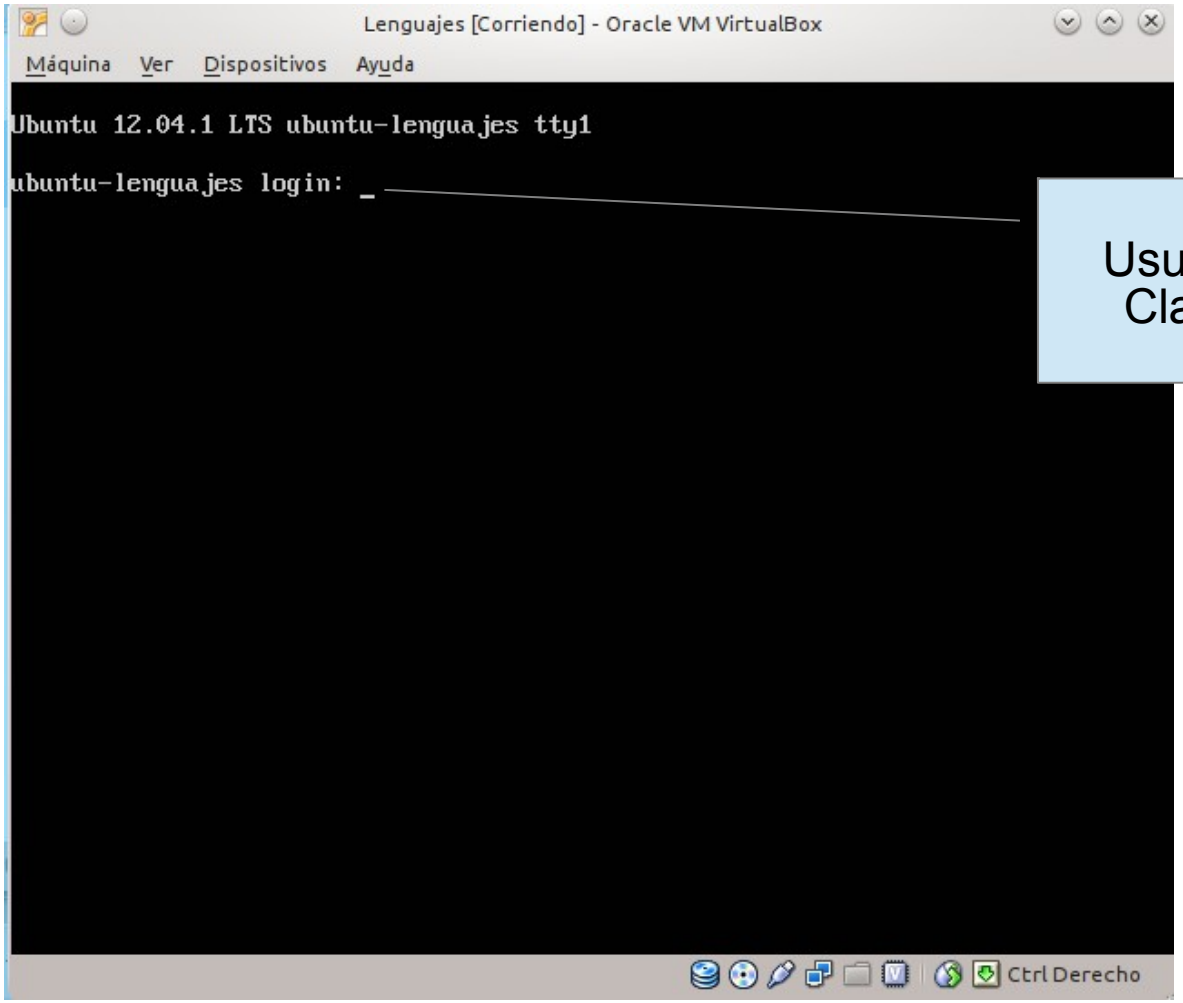
Instalación con VirtualBox



Instalación con VirtualBox

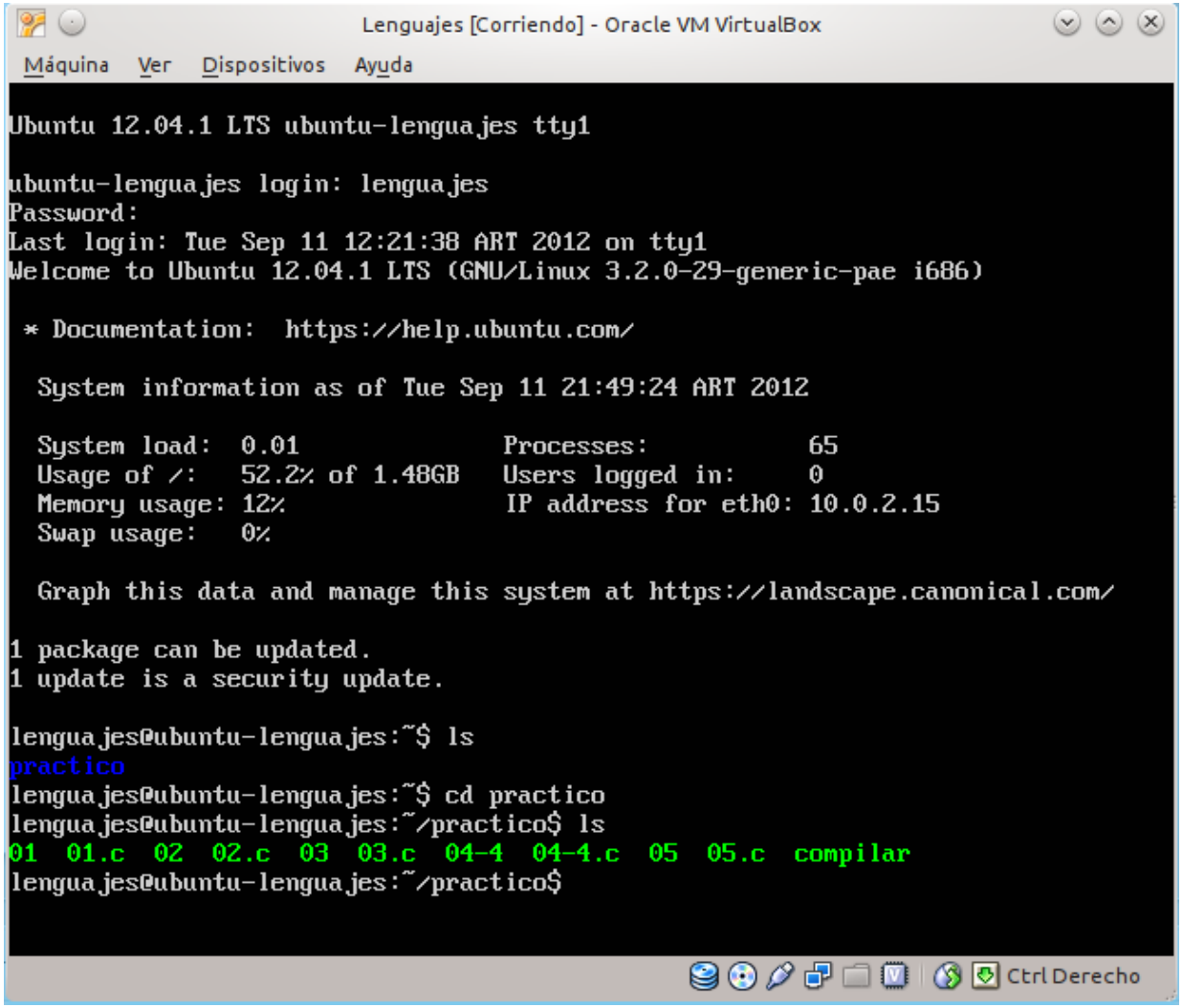


Instalación con VirtualBox



Usuario: lenguajes
Clave: lenguajes

Instalación con VirtualBox



Para mayor información

- Material disponible en el sitio web de la cátedra

FIN