Proceso Ligero

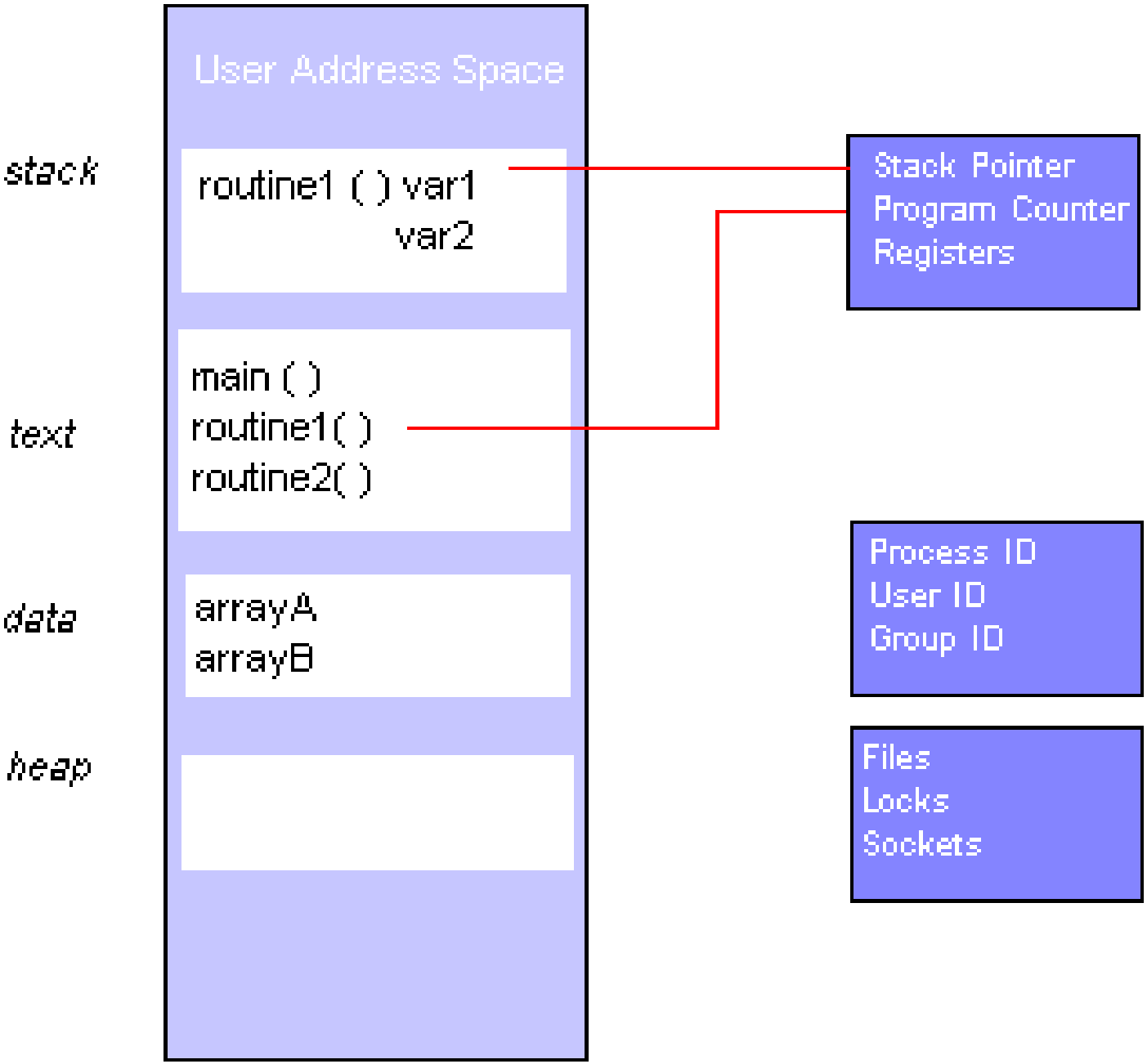
Un *thread* es un flujo de control perteneciente a un proceso.
Se les suele llamar también procesos ligeros, hebras o hilos

La sobrecarga debida a su creación y comunicación es menor que en los procesos pesados (fork)

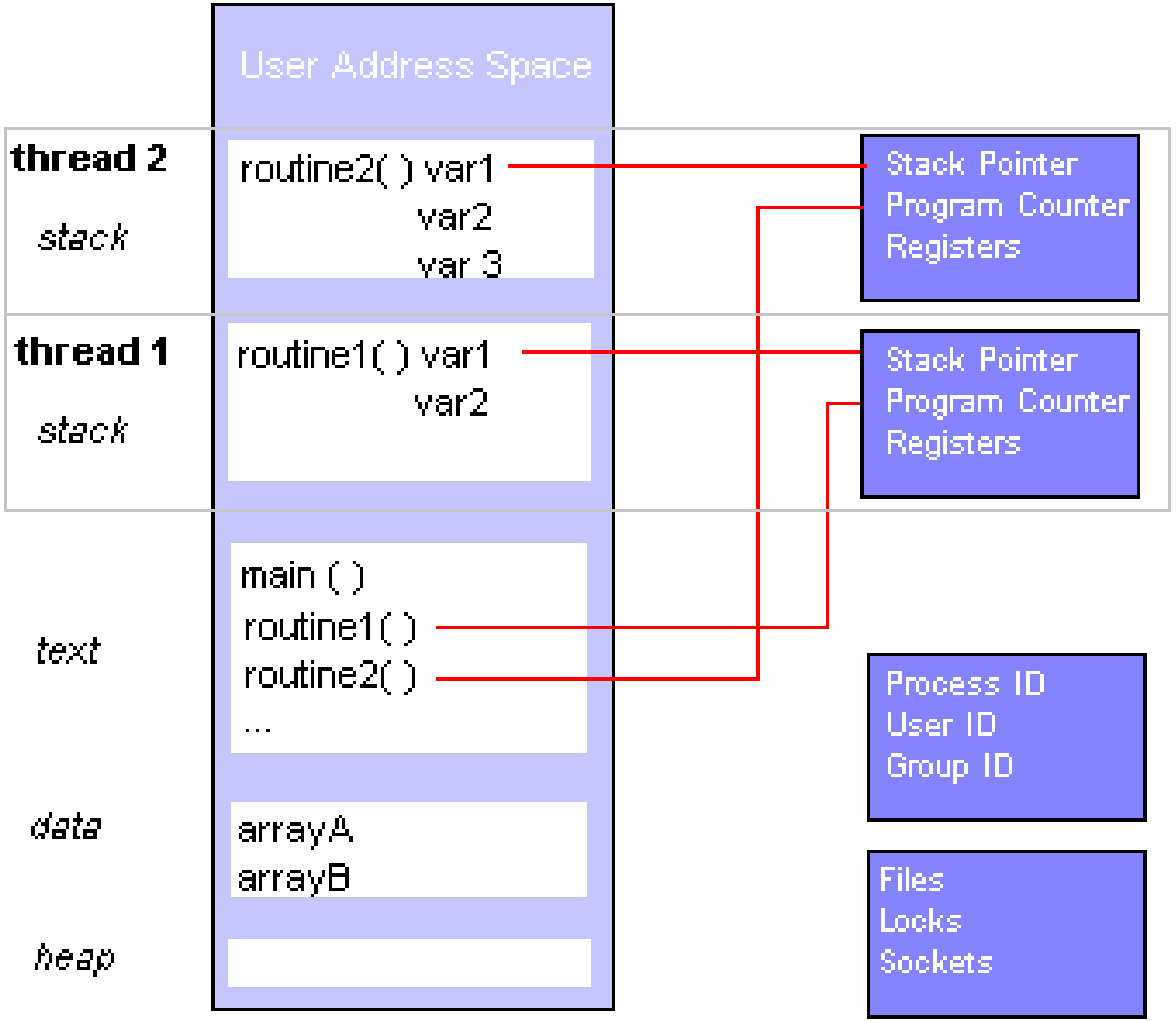
Cada hilo pertenece a un proceso pesado

Todos los hilos comparten su espacio de direccionamiento.
Sólo hay una copia de las variables

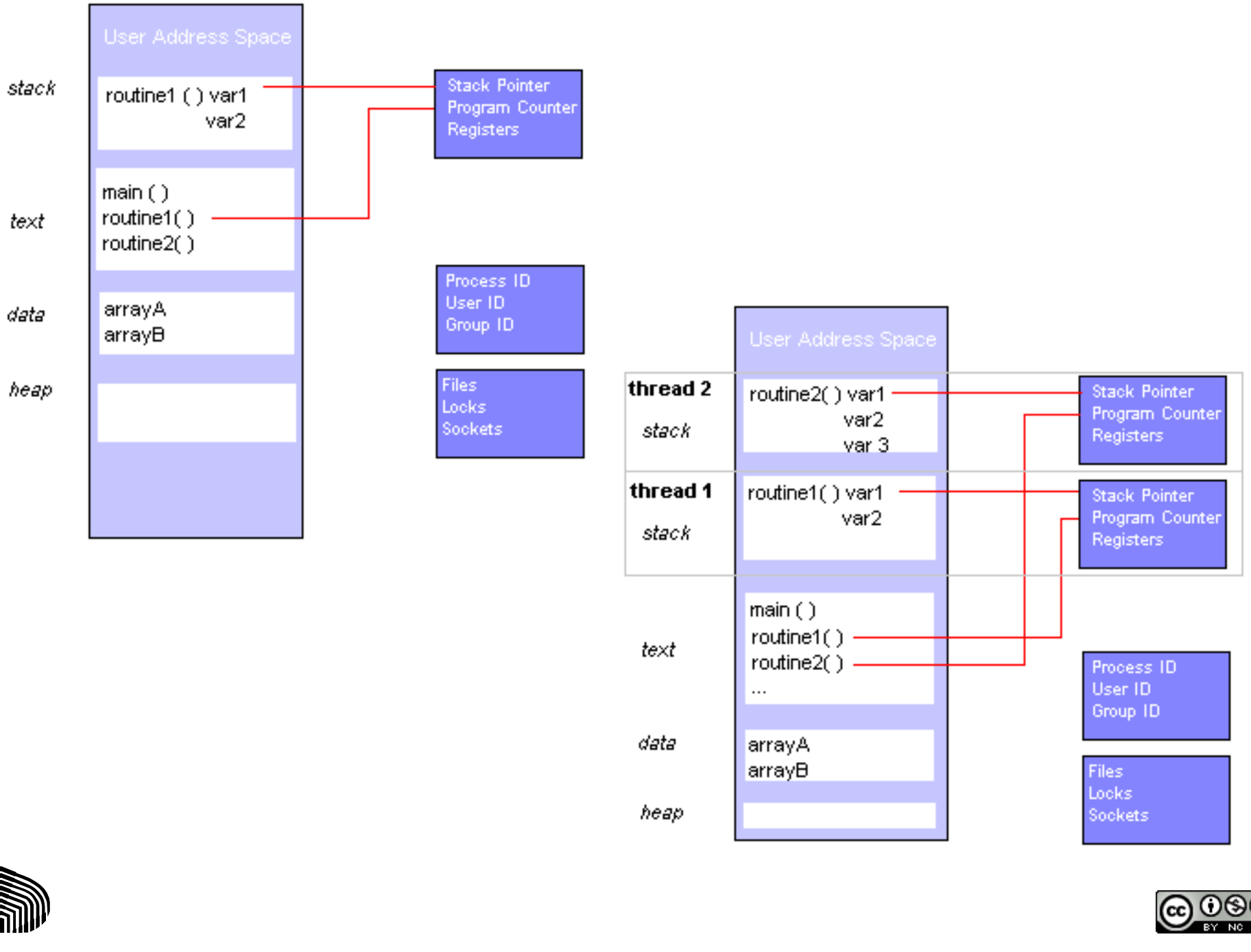
Prof. Angela Di Serio



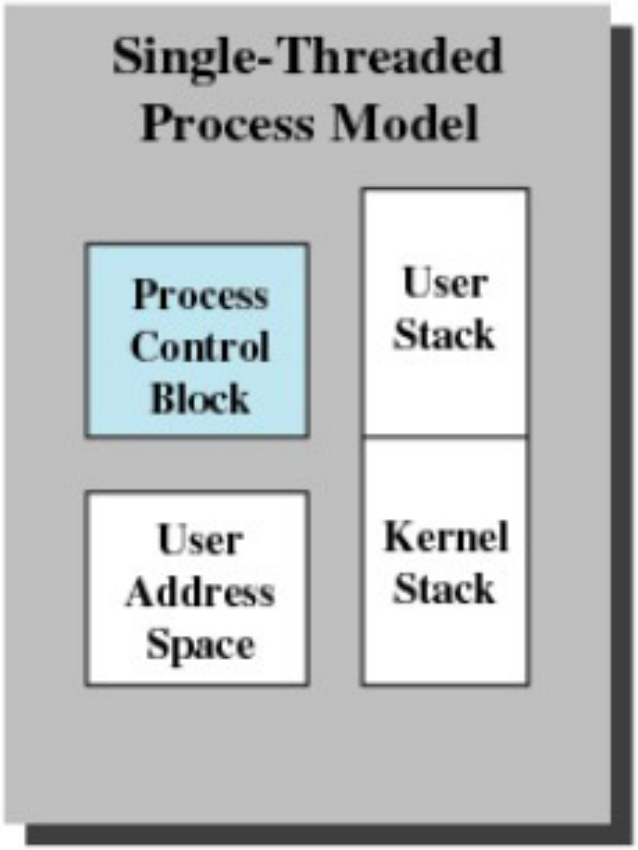
Prof. Angela Di Serio

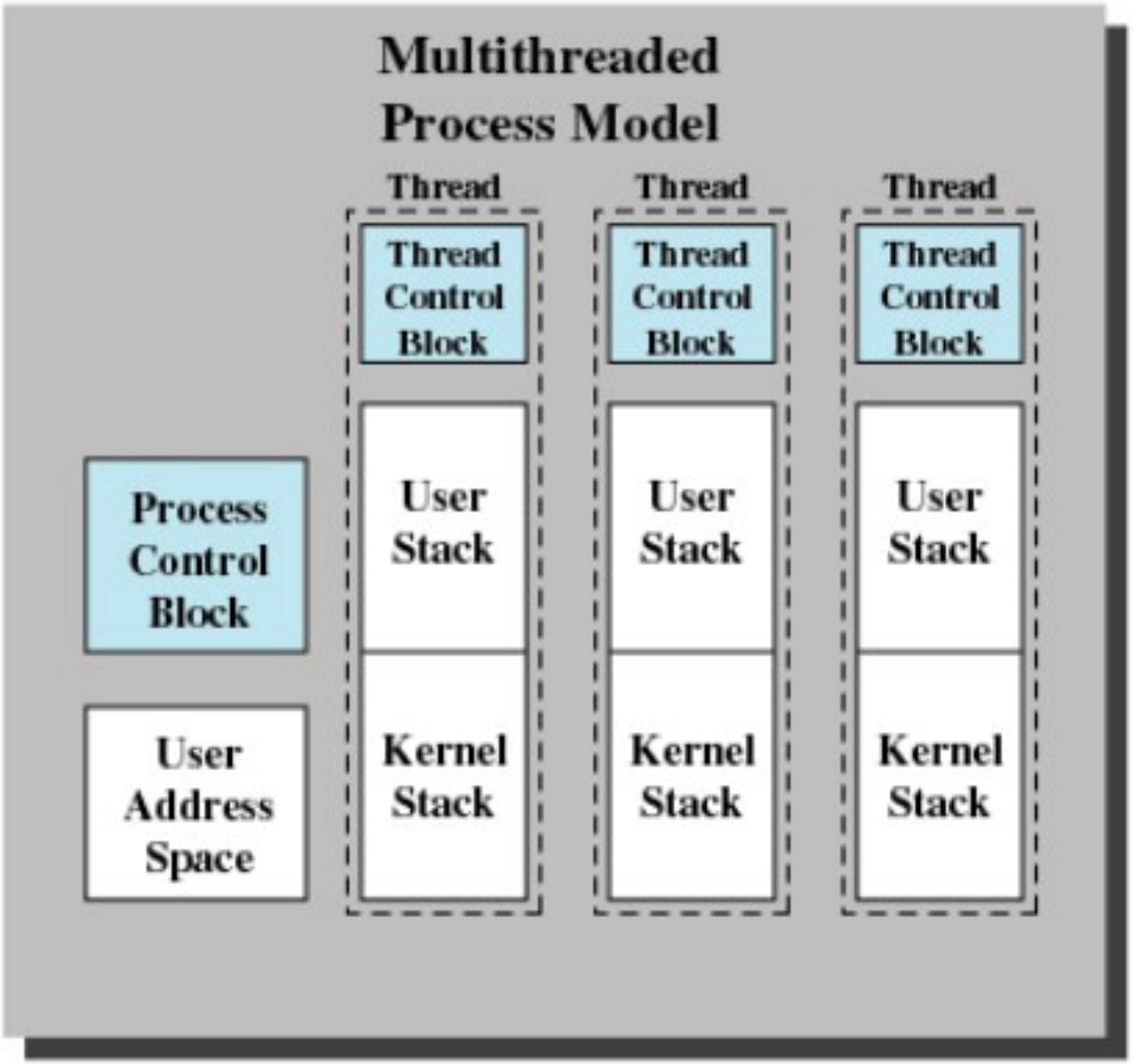


Prof. Angela Di Serio



Prof. Angela Di Serio





Prof. Angela Di Serio

Un proceso puede tener varios hilos y todos comparten los recursos del proceso y todos ejecutan dentro del mismo espacio de direcciones

Dado que comparten recursos, los cambios realizados por un hilo serán vistos por el resto de los hilos

Lectura y escritura a la misma dirección de memoria es posible en hilos, por lo tanto es necesario algún mecanismo de sincronización por parte del programador para evitar lo que se conoce como **Condición de Carrera**

Prof. Angela Di Serio

Beneficios

Mejor aprovechamiento de los recursos de procesadores en ambiente multiprocesadores/multicores

Mejor velocidad de comunicación entre hilos de un mismo proceso que entre procesos independientes del sistema

Dividir las aplicaciones en módulos funcionales bajo un mismo direccionamiento virtual

Prof. Angela Di Serio

Desventajas

Programación más difícil

Mayor dificultad de encontrar los errores

Limitación en el recurso de memoria

Prof. Angela Di Serio

Hilos a Nivel de Núcleo

Soportados a nivel del sistema operativo

El sistema operativo provee el soporte para la creación, planificación y administración de los hilos

El sistema reconoce cada hilo como una unidad de ejecución distinta a ser planificada.

Prof. Angela Di Serio

Hilos a Nivel de Núcleo

Beneficios

Mayor nivel de paralelismo en un sistema multiprocesador ya que varios hilos de un proceso pueden estar ejecutando en varios procesadores a la vez

Los hilos son independientes, por lo que al bloquearse un hilo de un proceso los demás hilos del proceso pueden seguir ejecutando

Prof. Angela Di Serio

Hilos a Nivel de Usuario

Son implementados a través de una biblioteca de usuario

La biblioteca provee soporte para la creación, planificación y administración de los hilos

El sistema operativo desconoce la existencia de estos hilos, por lo que solamente visualiza una unidad de ejecución

Prof. Angela Di Serio

Hilos a Nivel de Usuario

Beneficios

El cambio de contexto es menor comparado con los hilos a nivel de núcleo

Permite otro sistema de planificación ya que viene dado en la biblioteca de usuario

Prof. Angela Di Serio

Hilos POSIX

Librería

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

Algunos tipos y macros:

```
pthread_t idh;              //id del hilo
pthread_attr_t  atrh;       //atributos del hilo
PTHREAD_SCOPE_SYSTEM: hilo a nivel de kernel
PTHRAD_SCOPE_PROCESS: hilo a nivel de usuario
```

Inicializar y destruir atributos

```
pthread_attr_init(pthread_att_t *attr);
pthread_attr_destroy(pthread_att_t *attr);
```

Prof. Angela Di Serio

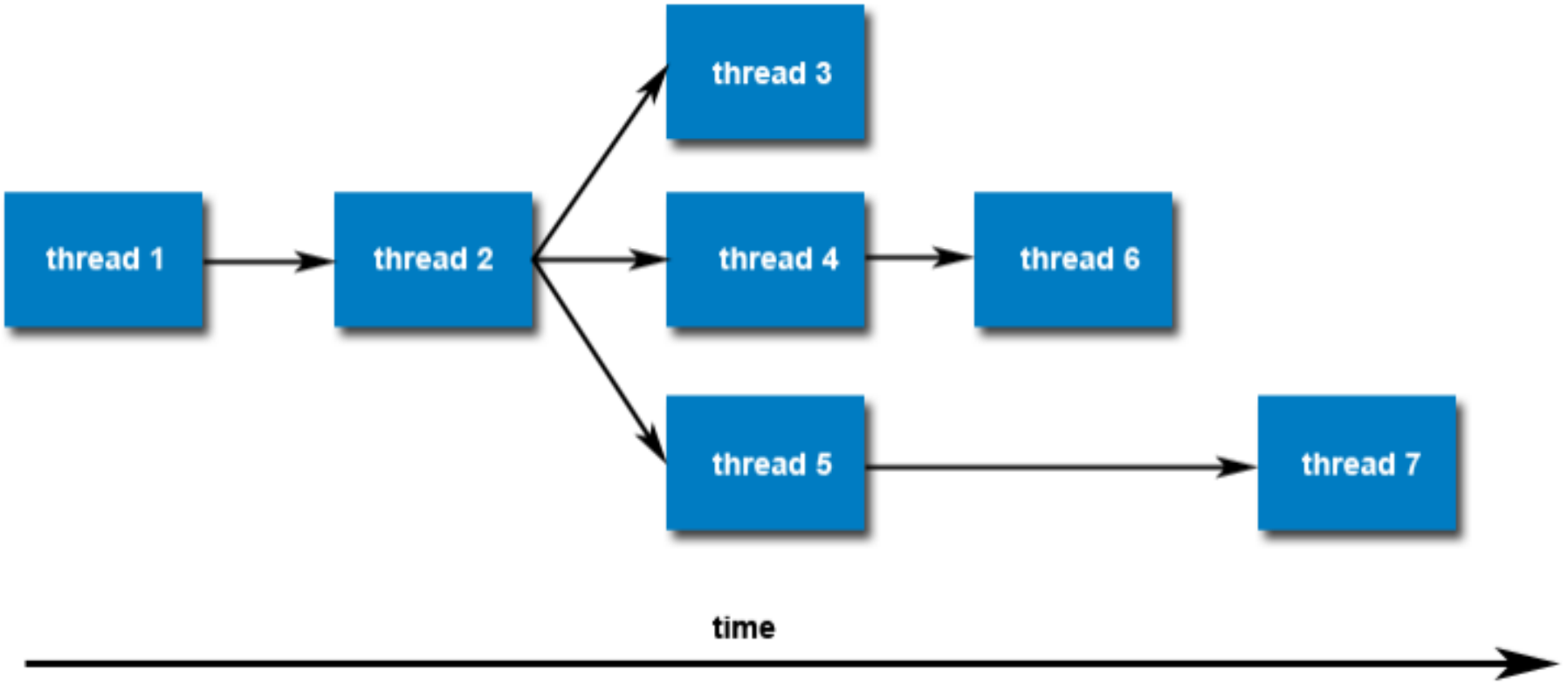
Función de creación **pthread_create**

Recibe como parámetro un apuntador a una función que contiene el código que ejecutará el hilo

La función a ejecutar tiene una firma o prototipo predefinido

La creación de hilos genera, dentro del proceso, una estructura de árbol

Prof. Angela Di Serio



Prof. Angela Di Serio

Función de creación **pthread_create**

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

pthread_t *	variable que referenciará al hilo creado
const pthread_attr_t *	parámetro de inicialización del hilo
void * (* función_thread) (void *)	apuntador a la función donde comenzará a ejecutar el hilo creado
void * arg	argumento para la función a ejecutar el hilo

Prof. Angela Di Serio

Función de creación **pthread_create**

El tipo de dato pthread_t permite definir hilos

Generalmente cuando se tiene un conjunto de hilos que realizan una misma tarea, se los agrupa en arreglos

```
pthread_t thrs[MAX_THREADS];

for (i = 0; i < MAX_THREADS; i++) {
    …
       if (rc = pthread_create(&thrs[i],….))
    …
}
```

Prof. Angela Di Serio

Función de creación **pthread_create**

Si el parámetro pthread_attr_t es nulo, el hilo será creado con valores por defecto

Atributo	Valor defecto	Resultado
Scope	PTHREAD_SCOPE_PROCESS	El thread es configurado como unbounded
Detachstate	PTHREAD_CREATE_JOINABLE	El valor de salida del thread es preservado luego que este termina
Stack size	1 MB	Tamaño del stack
Priority		El thread hereda la prioridad del padre

Prof. Angela Di Serio

Función de creación **pthread_create**

Se puede crear una variable de tipo

pthread_attr_t

para crear un conjunto de hilos con alguna característica particular

```
pthread_attr_t tattr;

pthread_attr_init(pthread_attr_t *tattr);
pthread_attr_init(&tattr);

pthread_attr_setstacksize((size_t)1024*1024*100);

pthread_create(&thr,&tattr,...);
```

Prof. Angela Di Serio

Función de creación **pthread_create**

Prototipo de la función a ejecutar el hilo

`void * func_thread (void *)`

El parámetro es de tipo void * lo que permite pasar cualquier tipo de datos a la función

```
struct persona {
   struct fecha nacimiento;
   int sexo;
   ... };
struct persona p;

pthread_create (…., func_thread, (void *) p)
```

Prof. Angela Di Serio

Ejemplo 1

```
main () {
   long *taskids[NUM_THREADS];
   pthreads_t threads[NUM_THREADS];
   int t;

   for(t=0; t<NUM_THREADS; t++){
      taskids[t] = (long *) malloc(sizeof(long));
      *taskids[t] = t;
      printf("Creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL,
                          PrintHello, (void *) taskids[t]);
    ...
   }
}
```

Prof. Angela Di Serio

Función de destrucción **pthread_exit**

Típicamente es ejecutada cuando un hilo finaliza su ejecución y no tiene nada más para realizar

El programador puede asignar un status a la terminación de forma que algún otro hilo (a través de la primitiva pthread_join) obtenga el valor de la ejecución

La función no elimina ningún recurso asignado al proceso que haya sido pedido por el hilo

Si el hilo main finaliza con un pthread_exit, todos los hilos que hayan sido creados permanecerán activos

Prof. Angela Di Serio

Función de destrucción **pthread_exit**

```
#include <pthread.h>

void pthread_exit(void *retval);
```

Prof. Angela Di Serio

Función de sincronización **pthread_join**

La función pthread_join permite a un hilo esperar por la finalización de otro hilo

La función pthread_join es bloqueante sobre el hilo que la invoca

No es posible esperar por varios hilos como la función wait de C

Prof. Angela Di Serio

Ejemplo 2

```
struct thread_data{
   int  thread_id;
   int  sum;
   char *message;
};


void *PrintHello(void *threadarg) {
    struct thread_data *my_data;
    int taskid, sum;
    char *hello_msg=(char *)malloc(sizeof(char)*MAX_CHAR);
     …..
    my_data = (struct thread_data *) threadarg;
    taskid = my_data->thread_id;
    sum = my_data->sum;
    hello_msg = my_data->message;
    ….
    pthread_exit(NULL);
}
```

Prof. Angela Di Serio

Ejemplo 2 (continuación)

```
int main (int argc, char *argv[]){
    pthreads_t threads[NUM_THREADS];
    int t, rc;
    void *status;
    for(t=0; t<NUM_THREADS; t++){
        ...
       thread_data_array[t].thread_id = t;
       thread_data_array[t].sum = sum;
       thread_data_array[t].message = messages[t];
       rc = pthread_create(&threads[t], NULL, PrintHello,
                           (void *) &thread_data_array[t]);
    }
     ….
    for(t=0; t<NUM_THREADS; t++) {
       rc = pthread_join(thread[t], &status);
       if (rc) {
          printf("ERROR; return code from pthread_join()
                  is %d\n", rc);
          exit(-1);
       }
       printf("Main: completed join with thread %ld having a
              status of %ld\n",t,(long)status);
    }
```

Prof. Angela Di Serio