Introducción al concepto de threads

Dr.. Roberto Gómez Cárdenas
DCC del ITESM-CEM
rogomez@itesm.mx
http://homepage.cem.itesm.mx/rogomez

Función monitoreo descriptor archivo

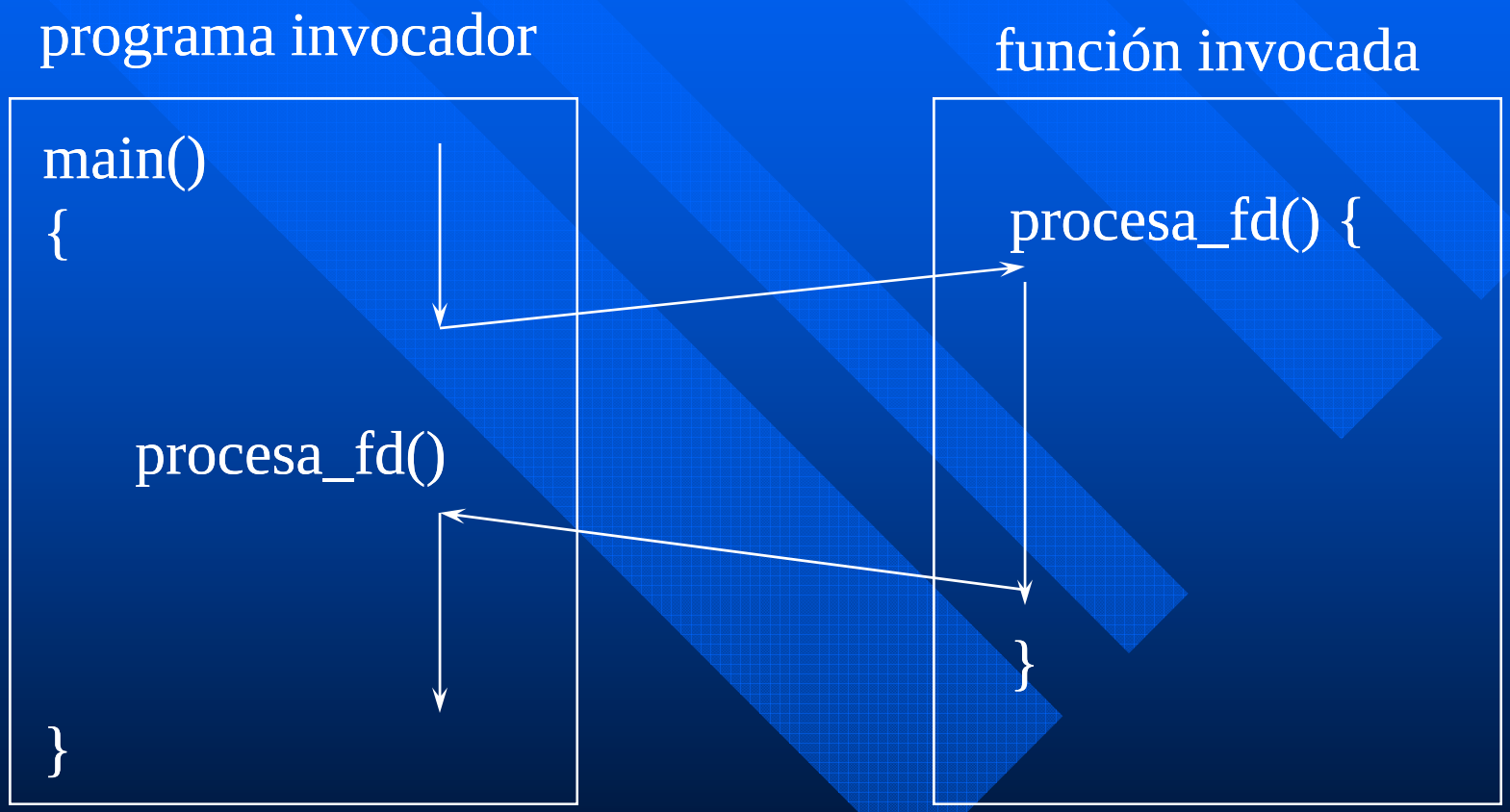
```
#include <sys/types.h>
#include<unistd.h>
#include<errno.h>
extern void  procesamiento_datos(char *, int);
void procesa_fd(int fd) {
   int nbytes;
   char buf[ BUFSIZE];
   for( ;  ;  )  {
        if  ( ( nbytes = read(fd, buf, BUFSIZE)) == -1) && (errno != EINTR))
            break;
        if ( !nbytes)
            break;
        procesamiento_datos(buf, nbytes);
   }
   return NULL;
}
```

Llamando función de monitoreo

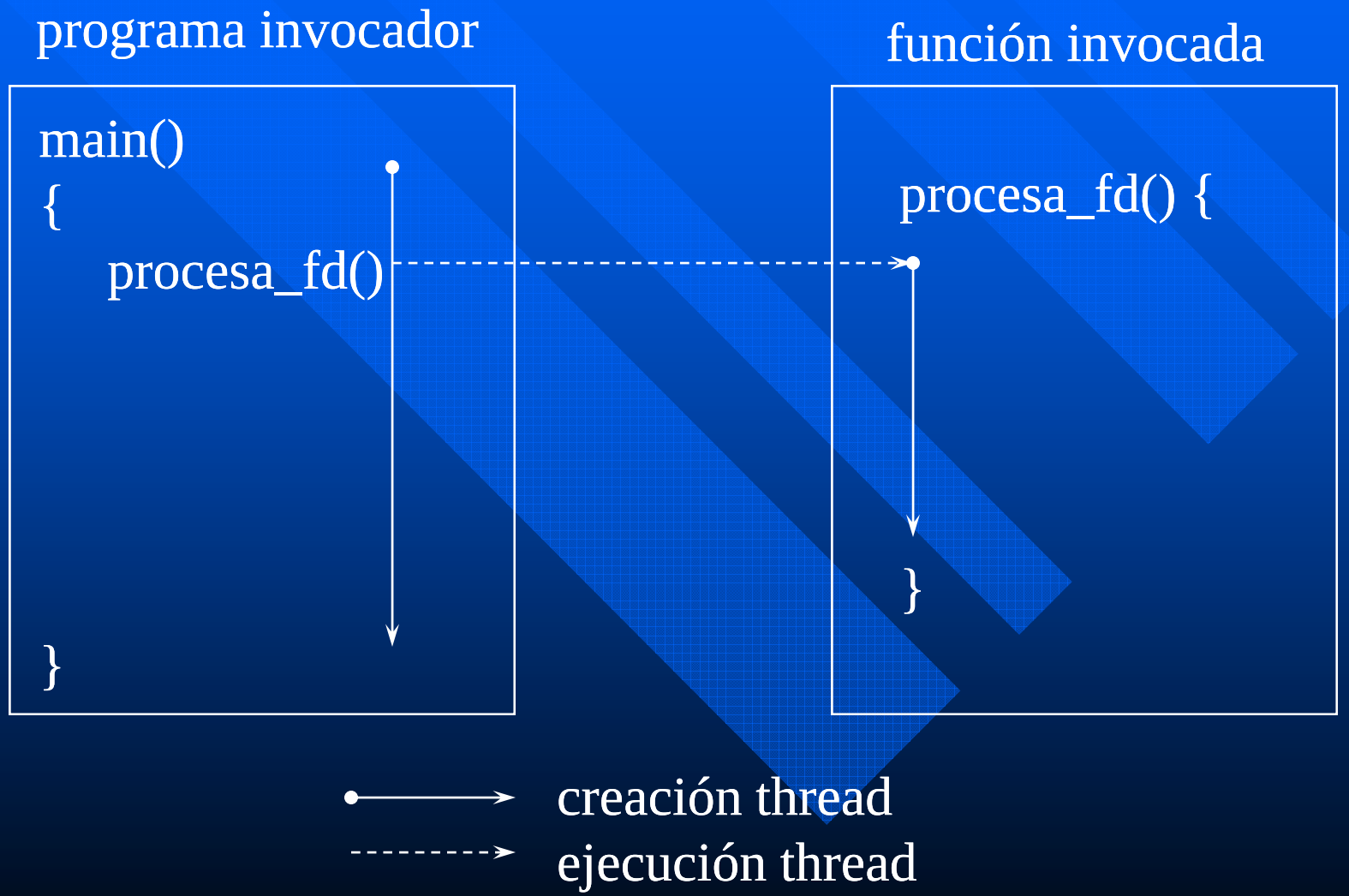
```
#include<stdio.h>
#include <sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<errno.h>
extern void procesa_fd(int fd)

main() {
   int fd;
   if (  (fd = open("toto", O_RDONLY) ) == -1)
            perror("Imposible abrir toto");
   else
            procesa_fd(&fd);
}
```

Secuencia de ejecución de llamada a la función



Una nueva secuencia ejecuta la función



¿Qué es un thread?

Un thread es una secuencia de pasos de ejecución en un programa. Cuenta con su propio contador de programa, así como un stack para llevar un control de variables locales y direcciones de regreso

Características de los threads

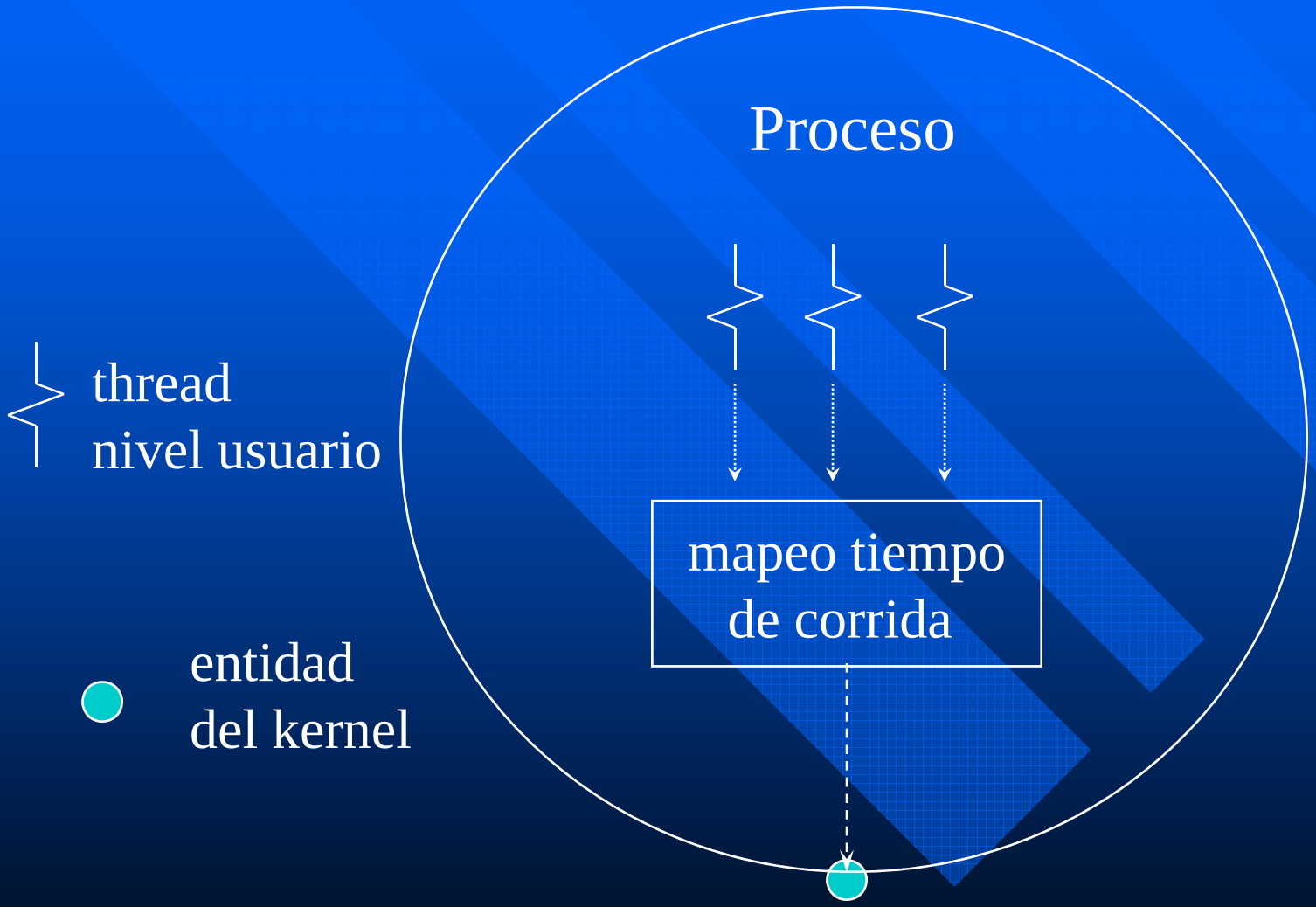
- Los threads se ejecutan independientemente y simultáneamente
- Comparten las instrucciones del proceso y casi todos sus datos
- Si un thread realiza un cambio en un dato en los datos compartidos este será percibido por otros threads en el mismo proceso

- Debido al comportamiento de información, su ejecución puede afectar al proceso de forma sorprendente
- Varios threads pueden proporcionar concurrencia dentro de un mismo proceso
- Si dos threads comparten recursos durante su ejecución, deben tener cuidado de que no se interfieran uno con el otro
- Un proceso tradicional de Unix solo tiene un thread de controlh

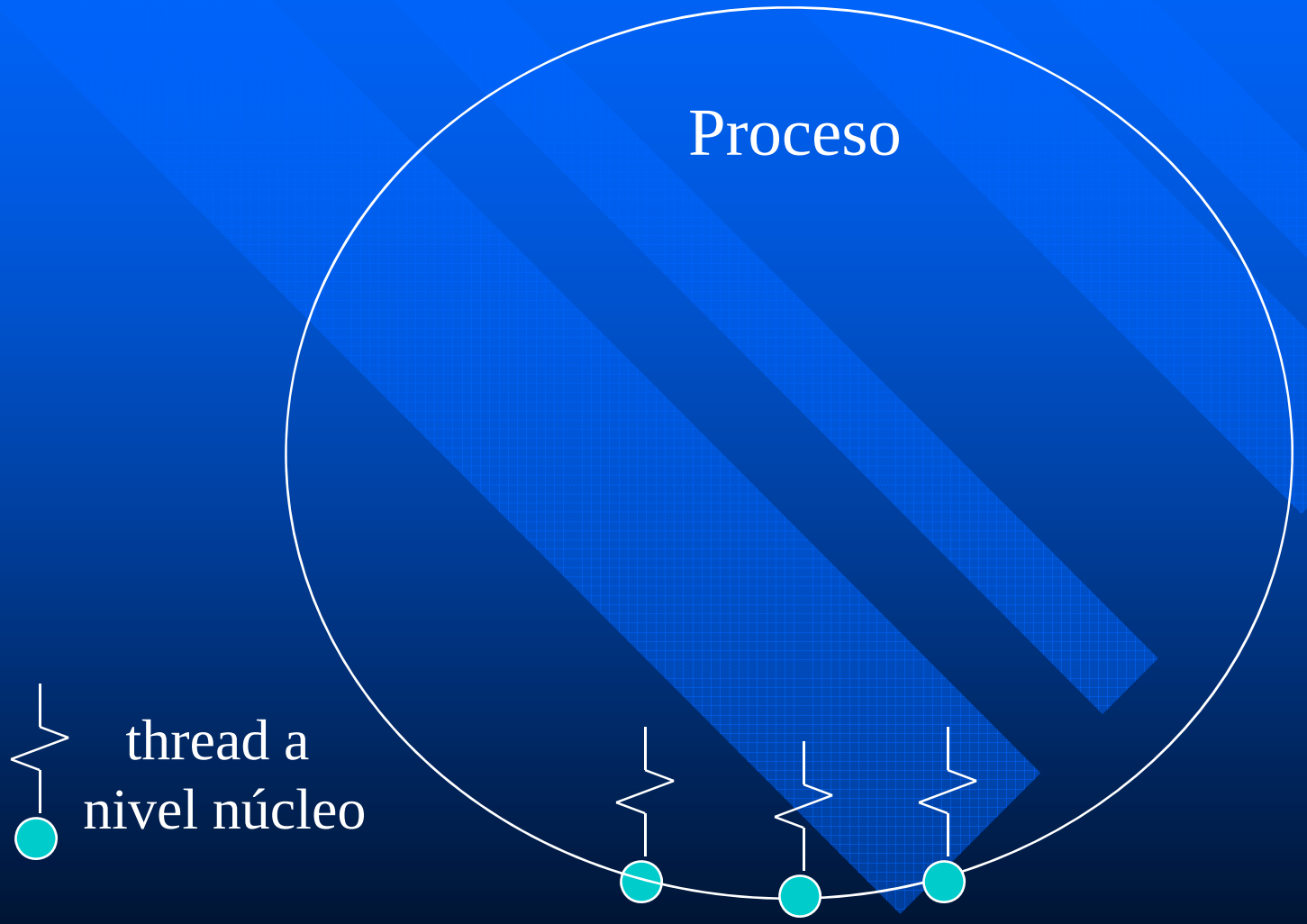
Modelos control threads

- Programa decide la siguiente instrucción en base al Program Counter
- Sistema operativo puede controlar la ejecución accediendo al Program Counter
- Los threads no pueden seguir el mismo principio, tres modelos:
  - threads a nivel usuario o no acotados
  - threads a nivel kernel o acotados
  - threads híbridos

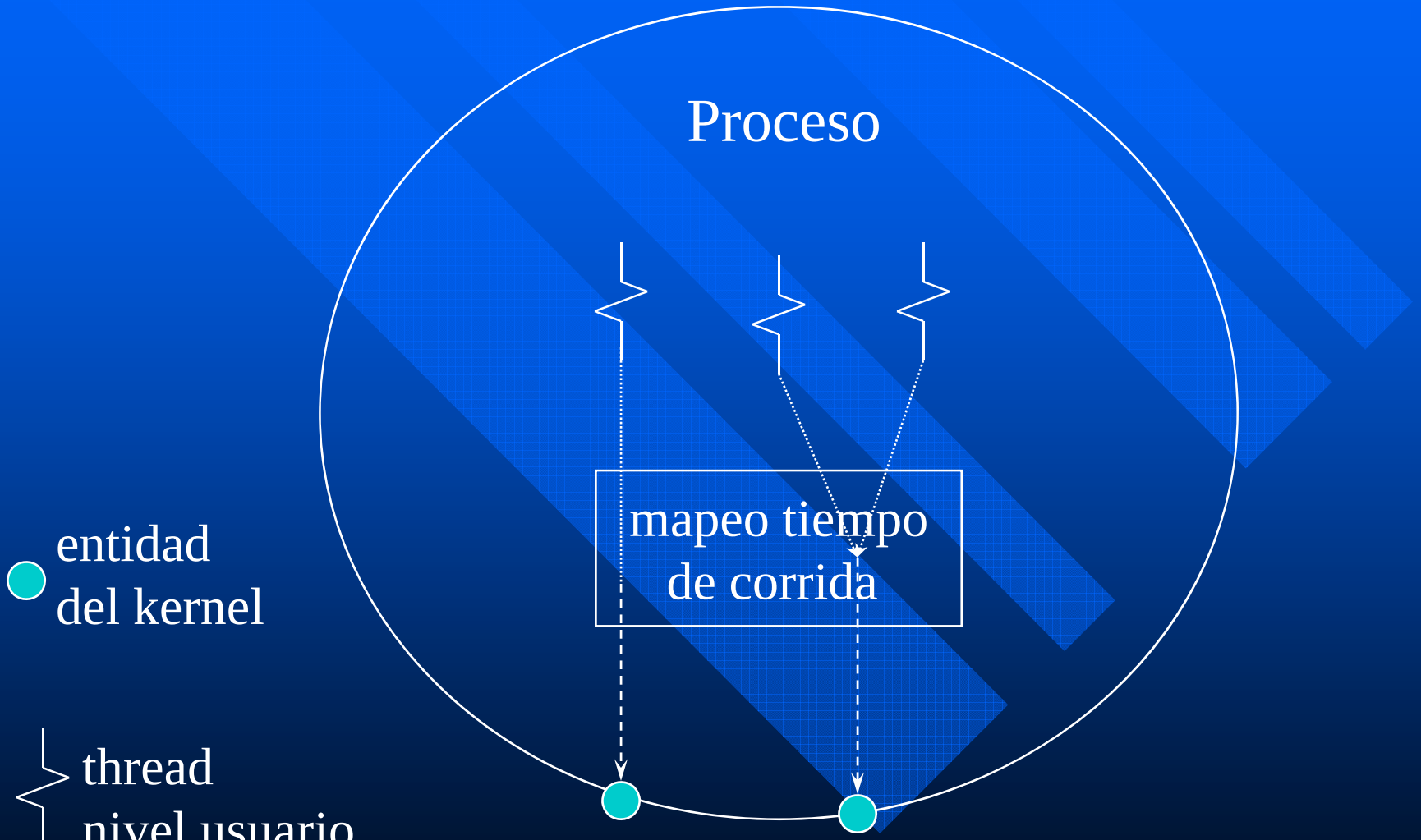
Threads nivel usuario



Threads nivel kernel



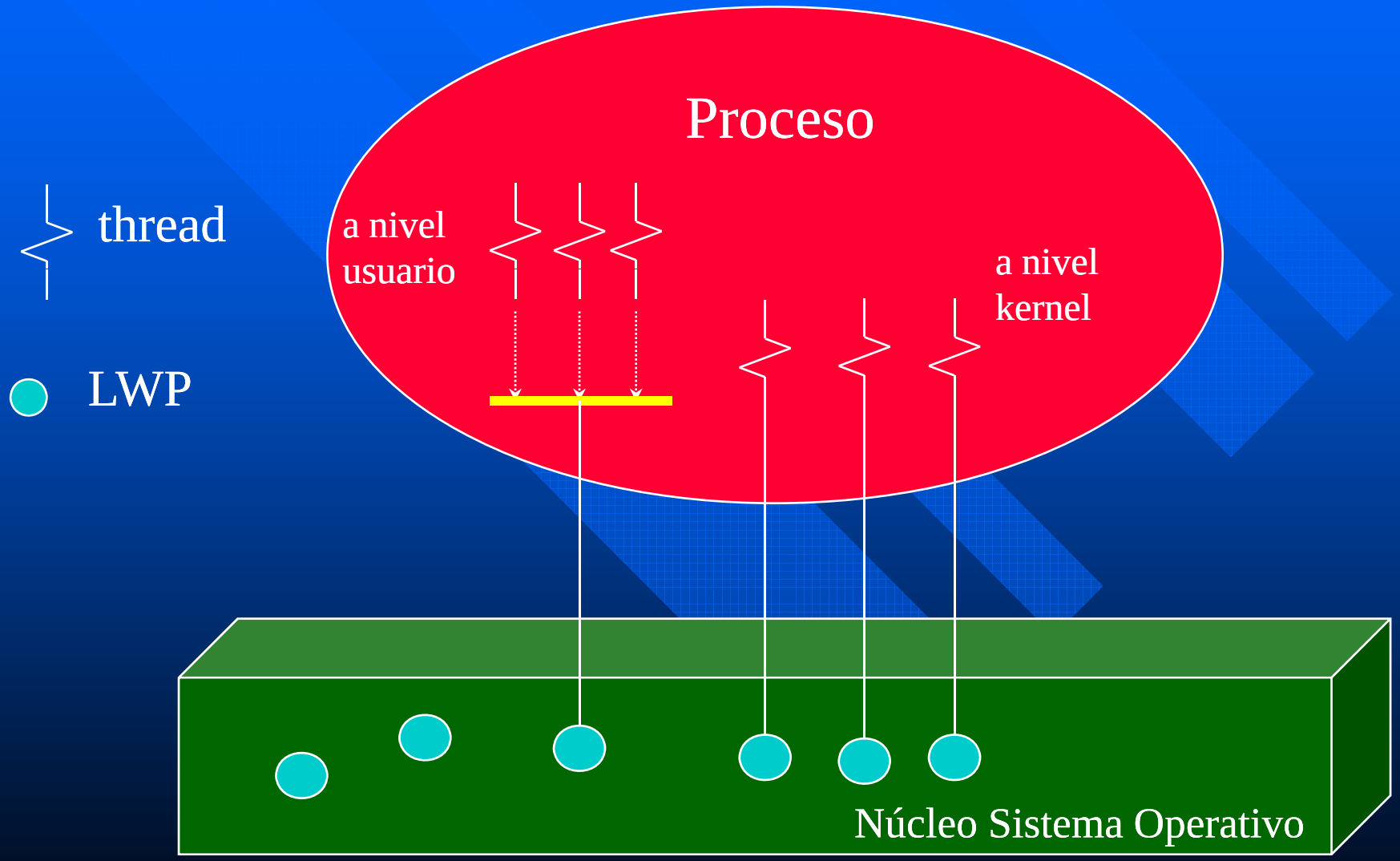
Modelo Híbrido



Procesos ligeros

- Las funciones relacionadas con los threads utilizan threads de control llamados LWP
- Un LWP se puede ver como una CPU virtual
- Mayoría programadores utilizan threads sin pensar en los LWP
- Funciones fopen() y fread() usan funciones open() y read()
- LWPs son el puente entre el nivel usuario y el nivel kernel
- Cada proceso cuenta con uno o más LWP, cada uno de los cuales ejecuta uno o más threads de nivel usuario

Procesos ligeros y threads



Estándares manejo threads

- Programación aplicaciones multithreads es de los 60’s
- Desarrollo en sistema Unix: mediados 80´s, después de la salida de Solaris 1
- Solaris proporciona un conjunto de funciones para el manejo de threads, conjuntados en librería *thread.h*
- POSIX (Portable Operating System Interface for uniX) desarrollo su conjunto de funciones agrupandolas en librería *pthread.h*

Llamadas Posix.1c y Sun Solaris 2

Descripción	POSIX	Solaris 2
Manejo Threads	pthread_create pthread_exit pthread_kill pthread_join pthread_self	thr_create thr_exit thr_kill thr_join thr_join
Exclusión Mutua	pthread_mutex_init pthread_mutex_destroy pthread_mutex_lock pthread_mutex_trylock pthread_mutex_unlock	mutex_init mutex_destroy mutex_lock mutex_trylock mutex_unlock

Descripción	POSIX	Solaris 2
Variables de condición	pthread_cond_init pthread_cond_destroy pthread_cond_wait pthread_cond_timedwait pthread_cond_signal pthread_cond_broadcast	cond_init cond_destroy cond_wait cond_timedwait cond_signal cond_broadcast

Creando threads

llamadas de sistema

Creación e inicialización threads

- Rutinas de creación e inicialización
- Se hace referencia a un thread a través de un identificador de tipo *pthread_t*
- Los threads comparten todo el espacio de direcciones del proceso donde son creados
- Un thread es ***dinámico*** si se puede crear en cualquier instante durante la ejecución del proceso y si no es necesario especificar por adelantado el número de threads

Creando un thread

*pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*rutina)(void *), void *arg);*

- crea un thread y lo pone en la cola de listos
- regresa 0 si todos salió bien, -1 en caso error
- *pthread_t *tid*: identificador del thread
- *const pthread_attr_t *attr*: atributos thread
- *void (*rutina)(void *)*: función llamada por el thread cuando este comienza ejecución
- *void *arg*: argumento de la función invocada

Ejemplo creación thread

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

main()
{
   pthread_t tid;
   int fd;
   if (  (fd = open("toto", O_RDONLY) ) == -1)
          perror("Imposible abrir toto");
   else
         if (pthread_create(&tid, NULL, procesa_fd, (void *)&fd) )
            perror("No se pudo crear el thread ");

                                  :
                                  :
```

Identificando al thread

pthread_self();

- regresa el identificador del thread que la llamo
- no requiere de ningún parámetro
- regresa un valor de tipo entero

Esperando al thread

*pthread_join(*pthread_t thread, void **value_ptr*);*

- suspende procesamiento hasta que thread termine
- thread debe ser un miembro del proceso actual
- el parámetro *value_ptr* contiene el valor del status de terminación del hilo esperado
- status disponible siempre y cuando el proceso esperado haya ejecutado un *pthread_exit()*

Terminando el thread

*pthread_exit(void *status);*

- termina la ejecución del thread que la invoca
- parámetro *status* disponible a través de la llamada de sistema *pthread_join()*
- puede llamarse desde cualquier lugar dentro del proceso