DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

Planificación de threads en POSIX

Mario Aldea
Michael González
Héctor Pérez

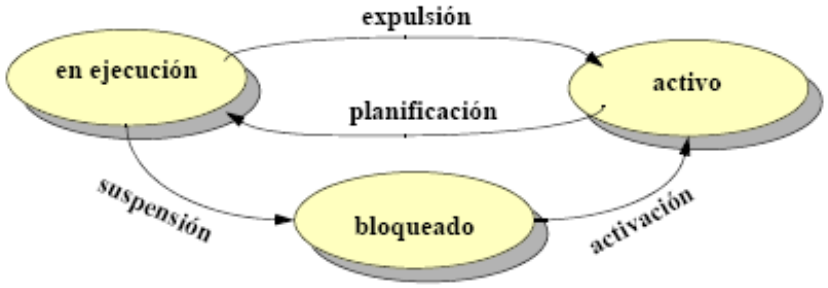
Planificación de threads en POSIX

- Conceptos básicos
- Políticas de planificación
- Interfaz para la planificación de threads
- Ejemplo

DSSE: M. Aldea, M. González y H. Pérez

Conceptos básicos

- Un thread puede estar en tres estados diferentes:
  - en *ejecución*: está en ejecución en un procesador
  - *activo*: está listo para ejecutar, pero aún no tiene un procesador disponible
  - *bloqueado o suspendido*: esperando a una condición para poder continuar ejecutando



DSSE: M. Aldea, M. González y H. Pérez

Conceptos básicos

- La **prioridad** representa un número entero positivo asociado a cada thread (o proceso)
  - es utilizado por la política de planificación
- La **política de planificación** representa un conjunto de reglas para determinar el orden de ejecución de los threads (o procesos)
  - afecta a la ordenación de los threads (o procesos) cuando:
    - se suspenden, bloquean o son expulsados
    - se activan
    - invocan una función que cambia la prioridad o la política
  - conceptualmente podemos considerar que existe una cola de threads activos por cada nivel de prioridad
    - y un thread ejecutando (fuera de la cola) por cada procesador

Políticas de planificación

- POSIX define las siguientes políticas de planificación
  - *SCHED_FIFO:* planificación expulsora por prioridad
    - sigue un orden FIFO para threads de la misma prioridad
  - *SCHED_RR:* planificación expulsora por prioridad
    - sigue un orden rotatorio para threads de la misma prioridad
    - cada thread tiene un slot temporal de ejecución predeterminado
  - *SCHED_SPORADIC:* planificación de servidor esporádico
  - *SCHED_OTHER:* otra política de planificación por prioridad
    - dependiente de la implementación

Planificación de threads

- POSIX define atributos de planificación
  - Son parte del objeto de atributos que se utiliza al crear el thread
  - POSIX no determina ningún valor por defecto en el sistema operativo
    - por tanto, debe configurarse siempre
  - Ámbito de contención ("*contentionscope*"):
    - **PTHREAD_SCOPE_SYSTEM**: ámbito de sistema
    - **PTHREAD_SCOPE_PROCESS**: ámbito de proceso
  - Herencia de atributos de planificación ("*inheritsched*")
    - **PTHREAD_INHERIT_SCHED**: hereda los del padre
    - **PTHREAD_EXPLICIT_SCHED**: usa los del objeto *attr*
  - Política de planificación ("*schedpolicy*")
    - **SCHED_FIFO**, **SCHED_RR**, **SCHED_SPORADIC** y **SCHED_OTHER**
  - Parámetros de planificación ("*schedparam*")
    - p.ej, el campo **sched_priority** para la prioridad

Interfaz de gestión de atributos (1/2)

```
#include <pthread.h>

int pthread_attr_getscope (const pthread_attr_t *attr,
                           int *contentionscope);

int pthread_attr_setscope (pthread_attr_t *attr,
                           int contentionscope);
```

PTHREAD_SCOPE_SYSTEM
PTHREAD_SCOPE_PROCESS

```
int pthread_attr_getinheritsched (const pthread_attr_t *attr,
                                  int *inheritsched);

int pthread_attr_setinheritsched (pthread_attr_t *attr,
                                  int inheritsched);
```

PTHREAD_INHERIT_SCHED
PTHREAD_EXPLICIT_SCHED

Interfaz de gestión de atributos (2/2)

```
int pthread_attr_getschedpolicy (const pthread_attr_t *attr,
                                 int *policy);

int pthread_attr_setschedpolicy (pthread_attr_t *attr,
                                 int policy);
```

SCHED_FIFO *SCHED_RR*
SCHED_SPORADIC *SCHED_OTHER*

```
int pthread_attr_getschedparam (const pthread_attr_t *attr,
                                struct sched_param *param);
int pthread_attr_setschedparam (pthread_attr_t *attr,
                                const struct sched_param *param);
```

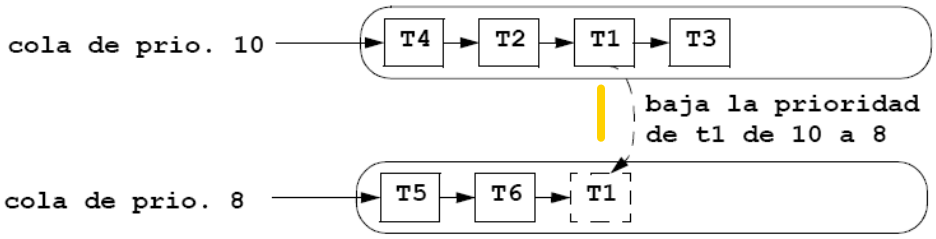
```
struct sched_param {
   int sched_priority;
   ...
}
```

Interfaz de cambio dinámico de atributos (1/2)

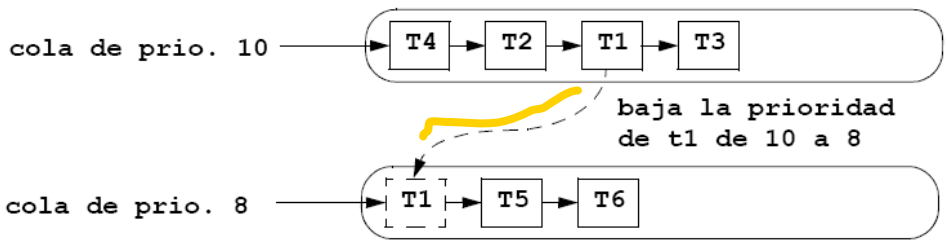
```
#include <pthread.h>

int pthread_getschedparam (pthread_t thread,
                           int *policy,
                           struct sched_param *param);

int pthread_setschedparam (pthread_t thread,
                           int policy,
                           const struct sched_param *param);


//Configurar los parámetros para un thread
int pthread_setschedprio(pthread_t thread, int prio);

//Configurar los parámetros y la política de planificación para un proceso
int sched_setscheduler (pid_t pid, int policy,
                        const struct sched_param *param);
```


Interfaz de cambio dinámico de atributos (2/2)

- El comportamiento de *pthread_setschedparam()* y de *pthread_setschedprio()* es diferente cuando se baja la prioridad del thread
  - pthread_setschedparam()



  - pthread_setschedprio()



Otras funciones

- Ceder el procesador:

```
int sched_yield (void);
```

- Leer los límites de los parámetros:

```
int sched_get_priority_max(int policy);

int sched_get_priority_min(int policy);

int sched_rr_get_interval(pid_t pid,
                          struct timespec *interval);
```


Planificación de threads: Ejemplo (1/4)

```
// Dado que se utiliza la librería pthread, debe indicarse en el
// proceso de compilación.
// Ejemplo: gcc main.c -lpthread
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <misc/error_checks.h>
#include <misc/timespec_operations.h>
#include <misc/load.h>

// datos transferidos a cada thread como argumento
struct thread_params {
  float execution_time;
  struct timespec period;
  int id; // identificador de la tarea
};
```

Planificación de threads: Ejemplo (2/4)

```
// Thread periódico:
// Pone un mensaje en pantalla, consume tiempo de CPU, y pone otro mensaje
void * periodic (void *arg) {
  struct thread_params params = *(struct thread_params*)arg;
  struct timespec next_time;

  // lee la hora de la primera activación de la tarea
  clock_gettime(CLOCK_MONOTONIC, &next_time);

  // lazo que se ejecuta periódicamente
  while (1) {
    printf("Thread %d empieza \n",params.id);
    eat(params.execution_time); // función para consumir tiempo de CPU
    printf("Thread %d acaba \n",params.id);

    // espera al próximo periodo
    incr_timespec(&next_time,&params.period); // sumo el periodo a la activación previa
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &next_time, NULL) );
  }
  pthread_exit(NULL);
}
```


Planificación de threads: Ejemplo (3/4)

```
// Programa principal que crea dos tareas con diferentes
// prioridades y periodos

int main() {
  pthread_attr_t attr;
  pthread_t t1,t2;
  struct thread_params t1_params, t2_params;
  struct sched_param sch_param;

  // Crea el objeto de atributos
  CHK( pthread_attr_init (&attr) );

  // Asigna cada atributo (no olvidar el atributo inheritsched)
  CHK( pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED) );

  CHK( pthread_attr_setschedpolicy(&attr, SCHED_FIFO) );

  sch_param.sched_priority = sched_get_priority_max(SCHED_FIFO)-5;
  CHK( pthread_attr_setschedparam(&attr,&sch_param) );
```

Planificación de threads: Ejemplo (4/4)

```
  // Prepara los argumentos del primer thread
  t1_params.period.tv_sec = 2; t1_params.period.tv_nsec = 0;
  t1_params.execution_time = 1.0;
  t1_params.id = 1;

  CHK( pthread_create(&t1,&attr,periodic,&t1_params) ); // crea el primer thread

  // Cambia la prio. en los atributos para crear el segundo thread
  sch_param.sched_priority = sched_get_priority_max(SCHED_FIFO)-6;
  CHK( pthread_attr_setschedparam(&attr,&sch_param) );

  // Prepara los argumentos del segundo thread
  t2_params.period.tv_sec = 7; t2_params.period.tv_nsec = 0;
  t2_params.execution_time = 4.0;
  t2_params.id = 2;

  CHK( pthread_create(&t2,&attr,periodic,&t2_params) ); // crea el segundo thread

  CHK( pthread_join(t1, NULL) ); // Permite a los threads ejecutar para siempre
  return 0;
}
```