Master en Computación

Programación Concurrente

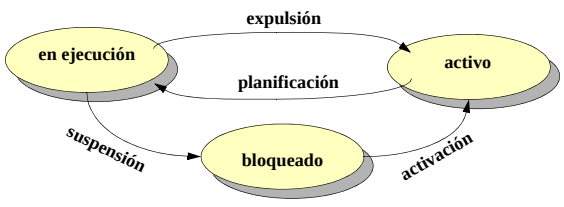
Bloque II: Programación concurrente en POSIX

Tema 5. Planificación de Threads

5.1 Introducción

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**



Conceptos básicos

***Prioridad*:**

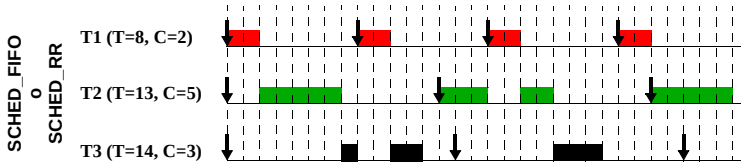
- **Número entero positivo asociado a cada thread (o proceso) y utilizado por la política de planificación**

***Política de Planificación*:**

- **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**
  - **llaman a 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**

5.2 Políticas de planificación

- **`SCHED_FIFO`: planificación expulsora por prioridad, con orden FIFO para la misma prioridad**
- **`SCHED_RR`: planificación expulsora por prioridad, con orden rotatorio para la misma prioridad; la rodaja temporal es fija**
- **`SCHED_SPORADIC`: planificación de servidor esporádico**
- **`SCHED_OTHER`: otra política de planificación por prioridad, dependiente de la implementación**



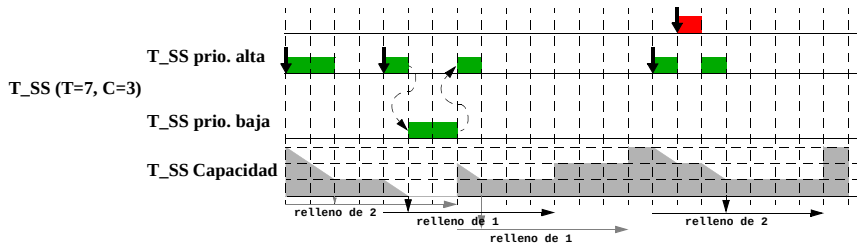
Política de servidor esporádico

Para procesar actividades aperiódicas a la prioridad deseada, con:

- **respuesta rápida a eventos**
- **efectos predecibles sobre tareas de prioridad inferior, incluso para ritmos de llegada no acotados**

Es una política de reserva de anchura de banda del procesador:

- **el tiempo de ejecución se limita a la capacidad de ejecución**
- **la capacidad se rellena después de un periodo de relleno**



5.3 Interfaz para la planificación de threads

Atributos de planificación:

- **Son parte del objeto de atributos que se utiliza al crear el thread**

- **Ámbito de contención ("*contentionscope*"); valores:**
  - **ámbito de sistema: `PTHREAD_SCOPE_SYSTEM`**
  - **ámbito de proceso: `PTHREAD_SCOPE_PROCESS`**

- **Herencia de atributos de planificación ("*inheritsched*"); *muy importante*, ya que si hay herencia no se hace caso del resto de atributos de planificación; valores:**
  - **hereda los del padre: `PTHREAD_INHERIT_SCHED`**
  - **usa los del objeto `attr`: `PTHREAD_EXPLICIT_SCHED`**
    - **valor por defecto en MaRTE OS**
    - **no tiene porqué ser el valor por defecto en otros SOs POSIX**

Atributos de planificación (cont.)

- **Política de planificación ("*schedpolicy*"); valores:**
  - **`SCHED_FIFO`, `SCHED_RR`, `SCHED_SPORADIC` y `SCHED_OTHER`**

- **Parámetros de planificación ("*schedparam*"):**

```
#include <sched.h>
#include <time.h>
struct sched_param {
    int sched_priority;
    int sched_ss_low_priority;
    struct timespec sched_ss_repl_period;
    struct timespec sched_ss_init_budget;
    int sched_ss_max_repl;
}
```

  - **Los cuatro últimos campos sólo se usan en la política de servidor esporádico**

Funciones para atributos de planificación

```
#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
```

Funciones para atributos de planificación(cont.)

```
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;
    int sched_ss_low_priority;
    struct timespec sched_ss_repl_period;
    struct timespec sched_ss_init_budget;
    int sched_ss_max_repl;
}
```

En MaRTE OS, si no se especifican atributos de planificación:
- política por defecto: `SCHED_FIFO`
- prioridad por defecto: intermedia para el thread main, 0 para los demás

Cambio dinámico de atributos de planificación

```
#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);
```

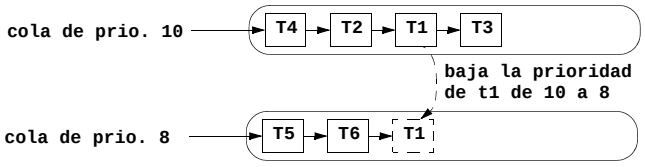
```
SCHED_FIFO
SCHED_RR
SCHED_SPORADIC
SCHED_OTHER
```

```
struct sched_param {
    int sched_priority;
    int sched_ss_low_priority;
    struct timespec sched_ss_repl_period;
    struct timespec sched_ss_init_budget;
    int sched_ss_max_repl;
}
```

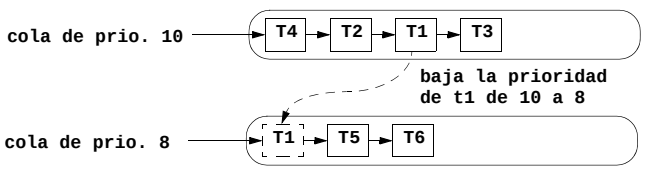
```
int pthread_setschedprio(pthread_t thread, int prio);
```


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);
```

Rodaja temporal en MaRTE OS: 0.03 seg.

5.4 Ejemplo: planificación de threads

"Come-tiempos" (`include/misc/load.h`):

```
/////////////////////////////////////////////////////
///                  LOAD.H                       ///
/////////////////////////////////////////////////////
//    La función eat() permite consumir el tiempo  //
//    de CPU indicado por for_seconds              //
//                                                 //
//    La función adjust() debe ser llamada una vez //
//    antes de llamar a eat(). No es necesario     //
//    llamarla posteriormente.                     //

void eat (float for_seconds);

void adjust (void);
```



```
// Programa principal

#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
};
```

```
// 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); // consume tiempo de CPU
    printf("Thread %d acaba \n",params.id);

    // espera al próximo periodo
    incr_timespec(&next_time,&params.period);
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                         &next_time, NULL) );
  }

  return NULL;
}
```



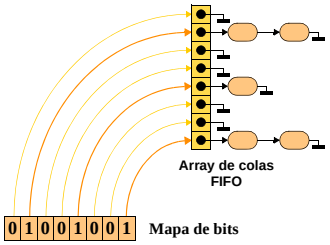
```
// Programa principal que crea dos tareas con diferentes
// prioridades y periodos

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

  adjust(); // requerido por la función eat()

  // 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) );
```



```
  // 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;

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

  // 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;

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

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

5.5 Implementación de la cola de tareas ejecutables en MaRTE OS

- **Optimizada para 20-50 threads**
- **Optimizada para pocos threads por prioridad (casi siempre n=1)**
- **Array de listas simplemente enlazadas y mapa de bits**



Operaciones sobre la cola:

- **Inserción: insertar a en la última posición de su lista de prioridad (O(n))**
- **Extracción: eliminar thread de su lista de prioridad (O(n))**
- **Búsqueda de thread más prioritario: búsqueda del primer bit a 1, el thread a la cabeza de esa lista es el más prioritario (O(1))**