Master en Computación

Plataformas de Tiempo Real

POSIX Avanzado y Extensiones

Tema 5. Planificación a nivel de aplicación

5.1 Motivación

Las políticas de planificación definidas en el POSIX no son suficientes para todos los tipos de aplicaciones

- **Las prioridades dinámicas permiten un mejor aprovechamiento de los recursos**
- **Los sistemas dinámicos (multimedia) requieren algoritmos de planificación flexibles**

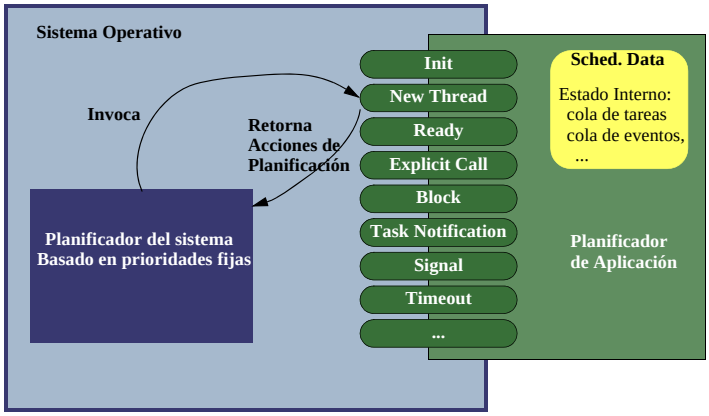
Existen muchas políticas basadas en prioridades dinámicas

- **Resultaría imposible estandarizarlas todas**

Nuestra propuesta al POSIX:

- **Interfaz que permita a las aplicaciones definir sus propios algoritmos de planificación**

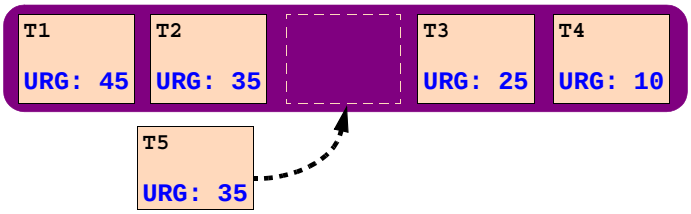
5.2 Descripción del Modelo: Estructura de un Planificador



Descripción del Modelo: Ordenación por "urgencia"

Noción abstracta de "urgencia" (un número entero):

- **Urgencia asignada a los threads por su planificador**
- **Permite mapear cualquier parámetro de planificación (plazo, holgura, valor, QoS, etc.)**
- **En cada cola de prioridad los threads se ordenan según el valor de su "urgencia"**



5.3 Creación de Planificadores de Aplicación

```
#include <sched.h>

int  posix_appsched_scheduler_create
   (const  posix_appsched_scheduler_ops_t *scheduler_ops,
    void * sched_data,
    int priority,
    posix_appsched_scheduler_id_t *sched_id);
```

- **`scheduler_ops`: operaciones del planificador**
- **`sched_data`: datos internos del planificador (serán pasados como parámetro a todas sus operaciones)**
- **`priority`: prioridad a la que se ejecutan las operaciones del planificador**
- **`sched_id`: identificador del planificador**

Estructura `posix_appsched_scheduler_ops_t`:

```
typedef struct {
  void (*init) (void * sched_data);
  void (*new_thread) (void * sched_data, pthread_t thread,
                      posix_appsched_actions_t * actions);
  void (*thread_ready) (void * sched_data, pthread_t thread,
                        posix_appsched_actions_t * actions);
  void (*explicit_call) (void * sched_data, pthread_t thread,
                         int user_event_code,
                         posix_appsched_actions_t * actions);
  void (*signal) (void * sched_data, siginfo_t siginfo,
                  posix_appsched_actions_t * actions);
  void (*notification_for_thread)(void * sched_data,
                              pthread_t thread,
                              posix_appsched_actions_t * actions);
  void (*timeout) (void * sched_data,
                   posix_appsched_actions_t * actions);
  ... // hay más operaciones primitivas
} posix_appsched_scheduler_ops_t;
```


5.4 Acciones de planificación

Desde sus operaciones, los planificadores pueden notificar al SO las acciones que desean realizar sobre sus tareas:

- **aceptar o rechazar una tarea (sólo desde `new_thread`)**
- **activar una tarea**
- **activar una tarea con el valor de urgencia deseado**
- **suspender una tarea**
- **activar una tarea en un instante futuro**
- **programar una notificación para un instante futuro**

Aceptación y rechazo de una tarea

La operación `new_thread` es invocada por el SO cuando una tarea del usuario solicita ser planificada por el planificador

El planificador puede aceptarla o rechazarla

- **en base a un test de planificabilidad, porque se ha alcanzado el número máximo de tareas, ...**

Aceptar de una tarea:

```
#include <pthread.h>

int posix_appsched_actions_addaccept
              (posix_appsched_actions_t *sched_actions,
               pthread_t thread);
```

Rechazar una tarea:

```
int posix_appsched_actions_addreject
              (posix_appsched_actions_t *sched_actions,
               pthread_t thread);
```

Activación y suspensión de una tarea

Activar una tarea (sin cambio de urgencia):

```
int posix_appsched_actions_addactivate
            (posix_appsched_actions_t *sched_actions,
             pthread_t thread);
```

Activar una tarea indicando su urgencia:

```
int posix_appsched_actions_addactivateurg
            (posix_appsched_actions_t *sched_actions,
             pthread_t thread,
             posix_appsched_urgency_t urgency);
```

- **si la tarea ya estaba activa, se cambia su urgencia**

Suspender una tarea:

```
int posix_appsched_actions_addsuspend
            (posix_appsched_actions_t *sched_actions,
             pthread_t thread);
```


Programación de acciones para el futuro

Programar la activación de una tarea para un instante futuro indicando su urgencia:

```
int posix_appsched_actions_addtimedactivation
      (posix_appsched_actions_t *sched_actions,
       pthread_t thread,
       posix_appsched_urgency_t urg,
       const struct timespec *at_time);
```

Programar la invocación de la operación primitiva `notification_for_thread` para un instante futuro:

```
int posix_appsched_actions_addthreadnotification
      (posix_appsched_actions_t *sched_actions,
       pthread_t thread,
       const struct timespec *at_time);
```

- **`at_time` basado en el `CLOCK_MONOTONIC`**

5.5 Creación de threads planificados

Se define una nueva política: `SCHED_APP`

Aparecen nuevos atributos:

- ***Planificador* de aplicación:**

```
int pthread_attr_setappscheduler
      (pthread_attr_t *attr,
       posix_appsched_scheduler_id_t scheduler);
```

- ***Parámetros de planificación específicos* de cada política de planificación:**

```
int pthread_attr_setappschedparam
      (pthread_attr_t *attr,
       void * param,
       size_t paramsize);
```

5.6 Invocación del planificador

Un thread planificado puede invocar explícitamente a su planificador:

```
int posix_appsched_invoke_scheduler
     (int user_event_code);
```

- **Puede utilizarse para notificar al planificador de algún cambio de estado de la tarea planificada**
- **Provoca la ejecución de la operación primitiva del planificador `explicit_call`**

5.7 Protocolos de Sincronización

Protocolo de Baker o "Stack Resource Protocol" (*SRP*) ("Techo de prioridad" aplicable a prioridades dinámicas)

- **+ Mejor tiempo de bloqueo de peor caso**
- **+ Aplicable a multiprocesadores**
- **+ Evita el bloqueo mutuo en monoprocesadores**
- **+ Evita cambios de contexto debidos a la sincronización**
- **- Necesario especificar el techo de cada recurso**

Herencia combinada de prioridad y urgencia

- **- No tiene ninguna de las propiedades anteriores pero...**
- **+ También acota el tiempo de bloqueo de peor caso**
- **+ No requiere indicar techos (bueno en sistemas dinámicos)**

Protocolo de Baker

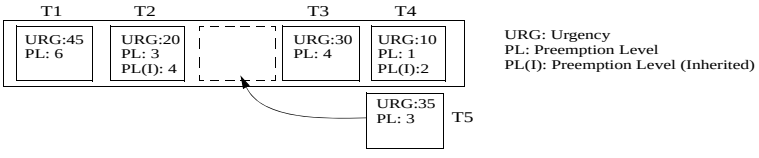
A cada tarea se asigna el nuevo atributo "Nivel de Expulsión"

- **Valor relativo a la capacidad de la tareas de expulsar a otras**
- **También se asigna a los mutexes (Techo)**

Nueva regla de planificación:

- **Una tarea no puede ejecutar hasta que su nivel de expulsión no sea estrictamente mayor que el techo del sistema**

Necesario combinar la regla con la ordenación por urgencia:



Herencia de Prioridad

Herencia de prioridad (Sha, Rajkumar y Lehoczky, 1990)

- **Inicialmente pensado para prioridades fijas**
- **Puede aplicarse a prioridades dinámicas puesto que...**
- **las demostraciones están basadas en el concepto de prioridad de cada "*job*"**

En nuestro modelo una tarea en posesión de un mutex hereda tanto la prioridad como la urgencia de las tareas que bloquea

5.8 Ejemplo: Planificador EDF

Primero pensamos qué es lo que debe hacer un planificador EDF:

- **Cuando se crea un thread**
  - **se calcula su primer plazo**
  - **se añade a la cola de tareas ejecutables en la posición que le corresponda según su plazo**
- **Cuando un thread acaba el trabajo correspondiente a su activación actual**
  - **se suspende hasta su próximo instante de activación**
- **Cuando llega el instante de activación de un thread**
  - **se calcula su nuevo plazo**
  - **se añade a la cola de tareas ejecutables en la posición que le corresponda según su plazo**

Ejemplo: Planificador EDF (cont.)

Diseñamos las correspondientes operaciones primitivas:

- **`new_thread()`:**
  - **invocada cuando un thread requiere ser planificado**
  - **el planificador anota periodo del thread y le activa**
- **`explicit_call()`:**
  - **invocada por las tareas planificadas cuando finalizan el trabajo correspondiente a la activación actual**
  - **El planificador programa una notificación temporizada para el próximo instante de activación del thread**
- **`notification_for_thread()`:**
  - **invocada al alcanzarse el instante de activación de un thread**
  - **el planificador calcula el nuevo plazo del thread y le activa**
- **`init()`:**
  - **inicializa las estructuras utilizadas por el planificador**

Ejemplo: Planificador EDF (cont.)

Se define la función:

```
posix_appsched_urgency_t URG (struct timespec ts);
```

- **retorna un valor de "urgencia" inversamente proporcional al valor de `ts` (a menor `ts` mayor urgencia)**

Ejemplo: Planificador EDF `new_thread()`

```
void edf_new_thread(edf_sched_data_t * sched_data,
                    pthread_t thread,
                    posix_appsched_actions_t *actions) {
  // Obtiene los parámetros EDF específicos del thread
  pthread_getappschedparam(thread, &param, &paramsize);
  // Anota su periodo y plazo
  data->period        = param.period;
  data->next_deadline = now + data->period;
  // Asocia los datos con el thread
  pthread_setspecific_for(sched_data->key, thread, data);
  // Acepta el thread
  posix_appsched_actions_addaccept(actions, thread);
  // Asigna la urgencia y activa el thread
  posix_appsched_actions_addactivateurg(actions, thread,
                                        URG(data->next_deadline));
}
```


Ejemplo: Planificador EDF `explicit_call()`

```
// invocada cuando el thread finaliza su trabajo actual
void edf_explicit_call(edf_sched_data_t * sched_data,
                       pthread_t thread,
                       int user_event_code,
                       posix_appsched_actions_t * actions) {
  // Obtiene los datos asociados con el thread
  pthread_getspecific_from(sched_data->key, thread, &data);
  // Suspende el thread
  posix_appsched_actions_addsuspend(actions, thread);
  // Programa una notificación para el próximo instante de
  // activación del thread
  posix_appsched_actions_addthreadnotification(actions,
                                               thread,
                                           &data->next_deadline);
}
```

Ejemplo: Planificador EDF `notification_for_thread()`

```
// invocada al alcanzarse el instante de activación de un thread
void edf_notification_for_thread(edf_sched_data_t * sched_data,
                                 pthread_t thread,
                                 posix_appsched_actions_t *actions) {

  // Obtiene los datos asociados con el thread
  pthread_getspecific_from(sched_data->key, thread, &data);

  // Calcula el nuevo plazo del thread
  data->next_deadline += data->period;

  // Activa el thread con su nuevo valor de urgencia
  posix_appsched_actions_addactivateurg(actions,
                                        thread,
                                        URG(data->next_deadline));
}
```


Ejemplo: Planificador EDF Creación del planificador

```
// por simplicidad se ha eliminado el chequeo de errores

// Crea una llave para almacenar datos específicos de los threads y
// la almacena en la estructura de datos del planificador
pthread_key_create(&edf_schr_data->key, NULL);

// Crea el planificador EDF
posix_appsched_scheduler_id_t edf_schr_id;
posix_appsched_scheduler_ops_t sched_ops =
    {.init = edf_init,
     .new_thread = edf_new_thread,
     .explicit_call = edf_explicit_call,
     .notification_for_thread = edf_notification_for_thread};
posix_appsched_scheduler_create(&sched_ops, &edf_schr_data,
                                THREAD_PRIO+1, &edf_schr_id);
```


Ejemplo: Planificador EDF Creación de thread y mutex

```
// Crea un thread EDF
pthread_attr_setschedpolicy(&attr, SCHED_APP);
pthread_attr_setappscheduler(&attr, edf_schr_id);
edf_param.period = 2.0;
pthread_attr_setappschedparam(&attr, &edf_param, sizeof(edf_param));
sch_param.sched_priority = THREAD_PRIO;
pthread_attr_setschedparam(&attr, &sch_param);
pthread_create(&th1, &attr, edf_thread_body, NULL);


// Crea un mutex con el protocolo de herencia
pthread_mutexattr_setprotocol(&mutexattr, PTHREAD_PRIO_INHERIT);
pthread_mutex_init(&mutex, &mutexattr);
```

Ejemplo: Planificador EDF
Thread EDF

```
void * edf_thread_body(void * arg)
{
  while (1) {
    // hace el trabajo útil (puede usar mutexes de protocolo
    // PTHREAD_PRIO_INHERIT para sincronizarse con otros threads)
    ...;

    // Fin del trabajo correspondiente la activación actual
    // y espera a la siguiente activación
    posix_appsched_invoke_scheduler(0);
  }
  return NULL;
}
```