Master en Computación

Plataformas de Tiempo Real

POSIX Avanzado y Extensiones

Tema 4. Planificación EDF

4.1 Políticas de planificación en POSIX

El estándar POSIX define tres 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, dependiente de la implementación (en MaRTE es idéntica a `SCHED_FIFO`)**

Los sistemas operativos pueden implementar además otras políticas

En MaRTE OS:

- **`SCHED_EDF`: Earliest Deadline First**

4.2 Política de planificación EDF

La teoría muestra que el uso de prioridades dinámicas permite aprovechar mejor los recursos en algunos casos

EDF (Earliest Deadline First) es la política basada en prioridades dinámicas más popular

- ✓ **Relativamente simple**
- ✓ **Es óptima (en monoprocesador): si un conjunto de tareas es planificable con alguna política, también lo será con EDF**
- ✓ **Puede garantizar el cumplimiento de los plazos para una ocupación superior que con prioridades fijas (hasta el 100%)**

Inconvenientes comparada con las prioridades fijas

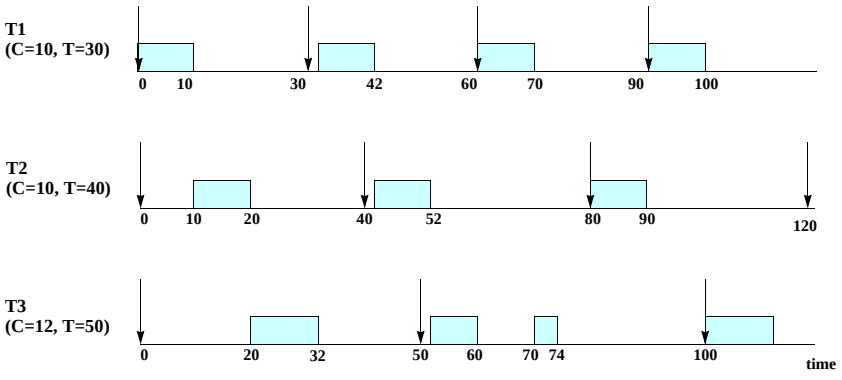
- × **Más compleja: mayor sobrecarga del sistema operativo**
- × **Inestable ante sobrecargas: cuando un thread sobrepasa su WCET no se sabe que thread perderá el plazo**

Política EDF

Definición

- **cada tarea tiene un nuevo atributo: su *plazo relativo***
- **en cada activación, el thread tiene una prioridad igual a su *plazo absoluto* (`instante de activación + plazo relativo`)**
- **para un mismo nivel de prioridad, las tareas se ordenan por su plazo absoluto (la tarea con plazo más corto se ejecuta primero)**

Ejemplo de planificación EDF



4.3 Integración con el resto de políticas POSIX

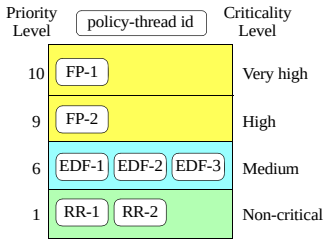
Es posible usar un esquema de planificación jerárquico

- **en que las prioridades fijas constituyen la política base**
- **las tareas de las distintas políticas deberían ocupar bandas de prioridad disjuntas**

Permite combinar en una aplicación las buenas propiedades de las diferentes políticas:

- **FIFO ⇨ predecibilidad para tareas críticas**
- **EDF ⇨ mejor aprovechamiento de los recursos para tareas no críticas**
- **RR ⇨ distribución equitativa de los recursos para tareas que no tengan requisitos de tiempo real**

Ejemplo de aplicación que combina diferentes políticas:



La ordenación por plazo sólo se aplica entre tareas de la misma prioridad

- **entre tareas de diferente prioridad siempre ejecuta la de mayor**

Cuando tareas EDF y FIFO comparten un mismo nivel de prioridad

- **las EDF se ejecutan antes**
- **es una situación que debería evitarse**

4.4 Recursos compartidos

En EDF puede darse un efecto similar a la inversión de prioridad no acotada

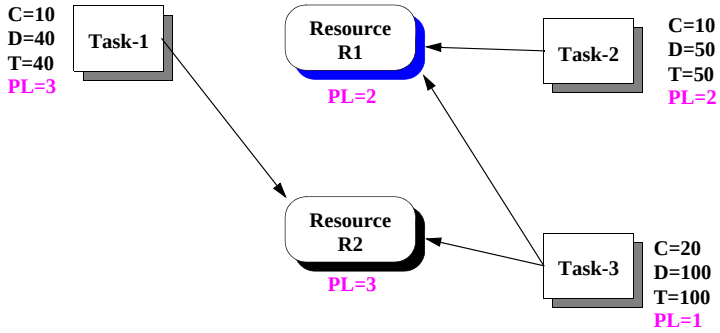
Para evitarlo se puede utilizar el *protocolo de Baker*, también conocido como "Stack Resource Protocol" (*SRP*)

- **el protocolo de techo de prioridad es un caso especial de este**
- **tienen las mismas buenas propiedades**
  - ✓ **minimiza la inversión de prioridad**
  - ✓ **asegura la exclusión mutua (no se necesita lock)**
  - ✓ **las tareas sólo pueden bloquearse al principio de su ejecución**
  - ✓ **una tarea sólo puede sufrir un bloqueo en cada activación**
  - ✓ **no pueden darse interbloqueos**

Asignación de "Preemption Levels"

El protocolo de Baker utiliza el "Preemption Level" (PL)

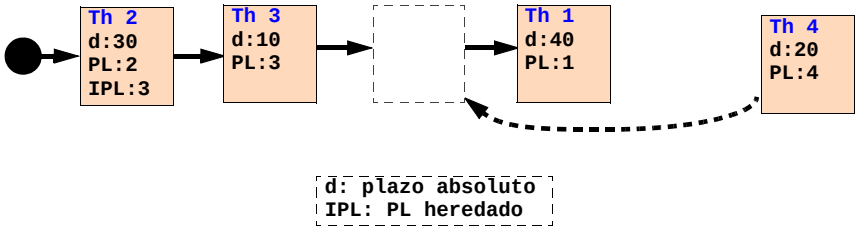
- **Número entero asignado a cada thread y cada mutex**
- **Threads: asignados en orden inverso a sus plazos relativos**
- **Mutexes: el mayor de los PLs de los threads que lo usan**



Nueva regla de ordenación de la cola de tareas ejecutables

Un thread recién activado se sitúa detrás de todos los threads que:

- **tengan un plazo menor que el suyo**
- **o tengan un PL heredado mayor o igual que su PL**
  - **una tarea hereda los PL de los mutexes que están en su poder**



DFP: Alternativa al SRP

"Deadline Floor inheritance Protocol" (DFP)

- **Recientemente propuesto por Alan Burns (2012)**
- **Implementado y probado en MaRTE OS**
  - **aún no incluido en la versión pública**

DFP es más sencillo y eficiente que SRP

Propuesto para su incorporación en el lenguaje Ada

- **reemplazando al SRP**
- **IRTAW-2013 (York, UK)**

DFP: Definición

DFP es estructuralmente equivalente al "protocolo de techo de prioridad" en prioridades fijas

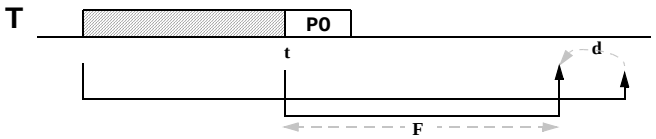
Cada recurso tiene un "Suelo de plazo" ("deadline floor")

- **el plazo relativo más corto de los threads que le usan**

Regla básica: el plazo absoluto de un thread puede acortarse mientras accede a un recurso:

```
d=min{d, t+F}
```

```
d: plazo absoluto
   del thread
F: "deadline floor"
   del recurso
t: instante de acceso
   al recurso
```



DFP tiene todas las buenas propiedades del SRP

- ✓ **minimiza la inversión de prioridad**
- ✓ **asegura la exclusión mutua (no se necesita lock)**
- ✓ **las tareas sólo pueden bloquearse al principio de su ejecución**
- ✓ **una tarea sólo puede sufrir un bloqueo en cada activación**
- ✓ **no pueden darse interbloqueos**
- **el tiempo de bloqueo es igual en ambos protocolos**

DFP no añade ninguna regla nueva a la planificación EDF

- **por lo que es mucho más fácil de implementar**

4.5 Interfaz para los threads (EDF&SRP)

Dos nuevos atributos: plazo relativo y "preemption level"

```
#include <pthread.h>

int pthread_attr_setreldeadline(pthread_attr_t *attr,
          const struct timespec *reldeadline);

int pthread_attr_getreldeadline(const pthread_attr_t *attr,
          struct timespec *reldeadline);


int pthread_attr_setpreemptionlevel(pthread_attr_t *attr,
              unsigned short int preemptionlevel);

int pthread_attr_getpreemptionlevel(const pthread_attr_t *attr,
              unsigned short int *preemptionlevel);
```

Cambio dinámico del plazo absoluto

Obtener el plazo absoluto actual del thread:

```
#include <pthread.h>
int pthread_getdeadline(pthread_t thread,
        clockid_t clock_id,
        struct timespec *deadline);
```

- El valor obtenido está medido en base al reloj indicado

Cambiar el plazo absoluto del thread:

```
#include <pthread.h>
int pthread_setdeadline(pthread_t thread,
     const struct timespec *deadline,
     clockid_t clock_id,
     int immediate);
```

- `deadline` es el nuevo plazo absoluto (basado en `clock_id`)
- si `immediate` es 1, el cambio se realiza inmediatamente
- si `immediate` es 0, el cambio se retrasa hasta la siguiente activación de la tarea

Esquema genérico de un thread EDF

```
void *edf_thread (void *arg)
{
  struct timespec next_activation, abs_deadline;

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

  // lazo que se ejecuta periódicamente
  while (1) {
    realiza la actividad periódica;

    // programa el nuevo plazo absoluto para la próxima activación
    // y espera
    incr_timespec(&next_activation, &period);
    add_timespec(&abs_deadline, &next_activation, &rel_deadline);
    CHK( pthread_setdeadline(pthread_self(), &abs_deadline,
                             CLOCK_MONOTONIC, 0) );
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                         &next_activation, NULL) );
  }
}
```


Cambio dinámico del “Preemption Level”

```
#include <pthread.h>
int pthread_setpreemptionlevel(pthread_t thread,
            short int preemptionlevel);

int pthread_getpreemptionlevel(pthread_t thread,
            short int *preemptionlevel);
```

4.6 Interfaz para los mutexes (EDF&SRP)

Como protocolo se usa el `PTHREAD_PRIO_PROTECT`

Nuevo atributo: "preemption level"

```
#include <pthread.h>

int pthread_mutexattr_setpreemptionlevel (pthread_mutexattr_t *attr,
                unsigned short int level);
int pthread_mutexattr_getpreemptionlevel (pthread_mutexattr_t *attr,
                unsigned short int *level);
```


Ejemplo de creación de mutex

```
CHK( pthread_mutexattr_init(&mutexattr) );
CHK( pthread_mutexattr_setprotocol(&mutexattr,
                    PTHREAD_PRIO_PROTECT) );
CHK( pthread_mutexattr_setprioceiling(&mutexattr,
                                      EDF_PRIO) );
CHK( pthread_mutexattr_setpreemptionlevel(
            &mutexattr, mutex_preemption_level) );
CHK( pthread_mutex_init(&mutex, &mutexattr) );
```