Bloque I: Principios de sistemas operativos

Notas:

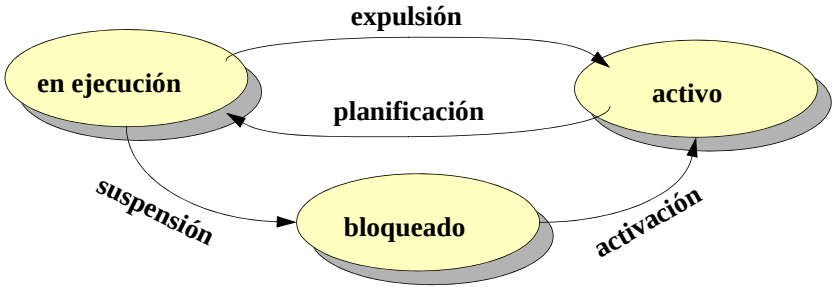
Tema 5: Planificación y despacho

- Conceptos.
- Políticas de planificación.
- Interfaz para planificación de procesos.
- Esquemas de planificación mixta entre procesos y *threads.*
- Planificación en sistemas multiprocesadores.
- Interfaz para planificación de *threads.*

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



Políticas de planificación: Conceptos básicos

Política de Planificación:

- Conjunto de reglas para determinar el orden de ejecución de los procesos o threads
- Afecta a la ordenación de los procesos o threads cuando:
  - se suspenden
  - son expulsados
  - se activan
  - llaman a una función que cambia la prioridad o la política

Conceptos básicos: Objetivos de la planificación

En sistemas de tiempo compartido

- *reparto equitativo* del procesador
- *eficiencia*: tiempos de respuesta promedio pequeños
- *potencia de cálculo*: procesar en promedio la mayor cantidad posible de actividades

En sistemas de tiempo real

- *predecibilidad*: tiempos de respuesta de peor caso acotados
- *urgencia:* tiempos de respuesta de peor caso pequeños, para actividades muy urgentes
- *criticalidad*: hacer primero las actividades críticas

Conceptos básicos (cont.)

Despacho:

- Es el conjunto de acciones que se llevan a cabo para comenzar la ejecución de una tarea

Expulsión:

- La planificación puede ser expulsora o no
- Si es expulsora, una tarea (que tenga mayor prioridad) puede interrumpir a otra y desalojarla de la CPU
- Si no es expulsora, cuando una tarea comienza continúa ejecutando hasta que voluntariamente cede la CPU
  - más sencilla
  - tiempos de respuesta mucho peores

Conceptos básicos (cont.)

***Prioridad*:**

- **Número asociado a cada proceso o thread y utilizado por la política de planificación**
- **A mayor valor, mayor prioridad**
- **prioridades *fijas*: número entero positivo que no cambia**
- **prioridades *dinámicas*: el valor cambia por sí solo**

***Plazo (Deadline)*:**

- **Instante de tiempo en el que una tarea debe haber finalizado su actividad**
- **Se usa como prioridad dinámica en la planificación EDF (*Earliest Deadline First*): a menor plazo, mayor prioridad**

2. Políticas de planificación

Cíclica o *round-robin*

- **Cada tarea recibe una "rodaja" del procesador**
- **Puede o no ser equitativa**

FIFO

- **Las tareas se ejecutan en el orden que llegan: no expulsable**

Basada en prioridades

- **Puede ser expulsable o no**

Primero el trabajo más corto

- **Para trabajos en lista (batch)**
- **Puede sufrir inanición**

Algunas políticas definidas en POSIX:

- **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_OTHER**: otra política de planificación por prioridad, dependiente de la implementación

Los parámetros de planificación definidos son:

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


3. Interfaz para la planificación de procesos

Fijar política y parámetros, o sólo parámetros:

```
#include <sched.h>
int sched_setscheduler (pid_t pid, int policy,
            const struct sched_param *param);
int sched_setparam (pid_t pid,
            const struct sched_param *param);
```

Leer política y leer parámetros:

```
int sched_getscheduler (pid_t pid);
     // devuelve la política
int sched_getparam (pid_t pid,
            struct sched_param *param);
```

Interfaz para la planificación de procesos (cont.)

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);
```


4. Planificación mixta: Ámbito de contención

Existen tres modelos de planificadores de threads:

- *de sistema*:
  - un planificador para todos los threads del sistema
  - los parámetros de planificación del proceso se ignoran
- *de proceso*:
  - un planificador para los procesos
  - otro planificador para los threads del proceso elegido
  - es el más eficiente, y menos predecible: no aconsejable para aplic. de tiempo real multi-proceso y multi-thread
- *mixto*:
  - un planificador para procesos y threads “globales”
  - otro para los threads “locales” del proceso elegido

5. Pl. en multiprocesadores: Dominio de asignación

Los threads están pensados para ser aplicables en multiprocesadores

La asignación de threads a procesadores es un campo de investigación abierto

POSIX define el dominio de asignación de un thread como el conjunto de procesadores en los que ese thread puede ser ejecutado

La forma de especificar el dominio de asignación es definida por la implementación

Las reglas de planificación sólo son deterministas para dominio de asignación de tamaño uno y estático

6. 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`

Atributos de planificación (cont.)

- Política de planificación (`schedpolicy`); valores:
  - `SCHED_FIFO`
  - `SCHED_RR`
  - `SCHED_OTHER`
- Parámetros de planificación(`schedparam`); es del tipo:

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


Funciones para atributos de planificación

```
#include <pthread.h>
int pthread_attr_setscope
     (pthread_attr_t *attr,
      int contentionscope);

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

int pthread_attr_setinheritsched
     (pthread_attr_t *attr,
      int inheritsched);

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

Funciones para atributos de planificación (cont.)

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

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

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

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


Funciones para el cambio dinámico de atrib. de planif.

```
#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);
```

Ejemplo: planificación de threads

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <string.h>
#include "load.h"

// datos transferidos a cada thread periódico

struct periodic_params {
  float period,execution_time;
  int id;
};
```


Ejemplo (cont.)

```
// Thread pseudo-periódico
// Pone un mensaje en pantalla, consume tiempo de CPU,
// y pone otro mensaje

void * periodic (void *arg)
{
  struct periodic_params params;

  params = *(struct periodic_params*)arg;
  while (1) {
    printf("Thread %d starts\n",params.id);
    eat(params.execution_time);
    printf("Thread %d ends\n",params.id);
    sleep ((int)params.period);
  }
}
```

Ejemplo (cont.)

```
// Programa principal, que crea dos threads periódicos
// con periodos y prioridades diferentes

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

  adjust();

  // Crea objeto de atributos
  if (pthread_attr_init (&attr) != 0) {
    printf("Error en inicialización de atributos\n");
    exit(1);
  }
```


Ejemplo (cont.)

```
  // Coloca cada atributo

  // El atributo inherisched es muy importante
  if (pthread_attr_setinheritsched (&attr,0) != 0) {
    printf("Error en atributo inheritsched\n");
    exit(1);
  }
  if (pthread_attr_setdetachstate
      (&attr,PTHREAD_CREATE_DETACHED) != 0) {
    printf("Error en atributo detachstate\n");
    exit(1);
  }
  if (pthread_attr_setschedpolicy (&attr, SCHED_FIFO) != 0) {
    printf("Error en atributo schedpolicy\n");
    exit(1);
  }
  sch_param.sched_priority =
    (sched_get_priority_max(SCHED_FIFO)-5);
```

Ejemplo (cont.)

```
  if (pthread_attr_setschedparam (&attr,&sch_param) != 0) {
    printf("Error en atributo schedparam\n");
    exit(1);
  }

  // Preparar el argumento del thread y crear el thread
  t1_params.period = 2.0;
  t1_params.execution_time = 1.0;
  t1_params.id = 1;
  if ((err=pthread_create(&t1,&attr,periodic,&t1_params)) != 0) {
    printf("Error en creacion del thread 1: %s\n",strerror(err));
    exit(1);
  }
```


Ejemplo (cont.)

```
  // Prepara atributos y parámetros para el segundo thread
  sch_param.sched_priority =
      (sched_get_priority_max(SCHED_FIFO)-6);
  if (pthread_attr_setschedparam (&attr,&sch_param) != 0) {
    printf("Error en atributo schedparam\n");
    exit(1);
  }
  t2_params.period = 7.0;
  t2_params.execution_time = 3.0;
  t2_params.id = 2;
  if ((err=pthread_create(&t2,&attr,periodic,&t2_params)) != 0) {
    printf("Error en creacion de thread 2: %s\n",strerror(err));
    exit(1);
  }

  // permite a los threads ejecutar un rato, y luego termina
  sleep (30);
  exit (0);
}
```