Sistemas Operativos
Práctica 4.

J. David Flores Peñaloza (dflorespenaloza@gmail.com)
Susana Hahn Martín Lunas (susuhahnml@ciencias.unam.mx)
A. Renato Zamudio Malagón (renatiux@gmail.com)

Fecha de entrega: Domingo 27 de Marzo 2016.

1 Objetivo

Modificar el calendarizador de prioridades para evitar la hambruna.

2 Introducción

Uno de los problemas que presentan los calendarizadores de prioridades es la hambruna. La hambruna se presenta cuando un thread de alta prioridad no termina de ejecutarse y por consecuencia los threads de menor prioridad nunca se ejecutan. Una solución a esto es hacer que los threads de alta prioridad que lleven mucho tiempo en ejecución disminuyan su prioridad, mientras que los de menor prioridad que tienen muy poco tiempo de ejecución aumenten su prioridad.

3 Requerimientos

Debes implementar un calendarizador avanzado que evite la hambruna, la prioridad de cada thread se debe recalcular cada cuatro ticks (esto pasa cuando se cumple *timer_ticks () % 4 == 0*) utilizando la siguiente fórmula:

*priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)*

La propiedad *nice* se debe definir por cada thread y se puede cambiar en cualquier momento,a variable *recent_cpu* también es por thread y se define como:

*recent_cpu = (2*load_avg)/(2*load_avg + 1) * recent_cpu + nice*

La fórmula anterior se debe recalcular para cada thread cuando se cumpla *timer_ticks () % TIMER_FREQ == 0*. El *recent_cpu* de un thread debe incrementarse en 1 si el thread se encuentra en ejecución al momento del *tick*. La fórmula para calcular *load_average* es la siguiente:

*load_avg = (59/60)*load_avg + (1/60)*ready_threads*

La variable ready_threads es el número de threads que están listos para ejecutar mas el thread en ejecución, el *load_average* se debe calcular cada que se cumpla *timer_ticks () % TIMER_FREQ == 0*.

Para poder hacer lo cálculos anteriores se necesitan operaciones de punto flotante, sin embargo, el kernel de Pintos descativa dichas operaciones por cuestiones de rendimiento (todos los sistemas operativos lo hacen). Por esa razón en necesario emular las operaciones de punto flotante con una biblioteca de punto fijo, dicha biblioteca se encuentra en el archivo *threads/fixpoint.h*.

Para probar que se están realizando de forma correcta los cálculos Pintos necesita que las siguientes funciones estén implementadas:

1. thread_set_nice
2. thread_get_nice
3. thread_get_load_avg
4. thread_get_recent_cpu

Las funciones 3 y 4 deben regrsar los valores de *load_average* y *recent_cpu* multiplicados por 100. Las pruebas que deben pasar son las siguientes:

1. mlfqs-load-1
2. mlfqs-load-60
3. mlfqs-load-avg
4. mlfqs-recent-1
5. mlfqs-block
6. mlfqs-fair-2
7. mlfqs-fair-20
8. mlfqs-nice-2

9. mlfqs-nice20

Para poder llevar a cabo la práctica es necesario tener implementado el canlendarizador de prioridades, en caso de que no lo tenga o que no confies en tu implementación, utiliza el sienguiente código fuente de Pintos para comenzar esta práctica:

https://dl.dropboxusercontent.com/u/98533171/pintos_mlfqs.tar.gz

4 Hints

1. Utiliza la biblioteca de punto fijo proporcionada.
2. Realiza los calculos de las variables en la función *thread_tick*.
3. Guarda las variables *recent_cpu*, *load_average* y *nice* en formato de punto fijo, y conviertelas a formato normal en las funciones ***get*** que acceden a ellas.
4. Utiliza variables globales en formato de punto fijo para almacenar las constantes de las formulas, incializalas en la declaración de las mismas. De esta forma solo realizarás el calculo de la conversión una vez.
5. Los calculos de *recent_cpu* y *priority* se deben de realizar para todos los threads, no solo los de la *ready_list*. Debes iterar sobre la lista *all_list*.
6. Recuerda reinsertar los threads cada que cambia su prioridad en la *ready_list*.
7. Utiliza la variable global *thread_mlfqs* para detectar que estas ejecutando las pruebas del calendarizador avanzado. Esto se debe hacer para que la implementación no interfiera con la pruebas anteriores.

Para mas información visita http://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html#SEC27

5 Entrega

Debes entregar un comprimido que tenga de nombre el número de cuenta de alguno de los integrantes del equipo. El comprimido debe contener un archivo README.txt con los nombres de los integrantes del equipo y solamente archivos que modificaste del src de pintos dejando intacta la structura de directorios. Por ejemplo si únicamente modificaste el archivo *timer.c* la estructura que debes entregar es:

```
|— README.txt
    |— src
    |— devices
```

|— thread.c

El comprimido se debe enviar por correo a la dirección *renatiux@gmail.com* con el asunto PRACTICA4