Pantilla de diseño

ISC 364

Proyecto: Procesos

Documento de Diseño

Miguel Estevez 2017-0200

Asuntos Preliminares

Citar aquí las fuente en linea o fuera de linea consultada para la preparación de su entrega, mas allá de la documentación de Pintos, libro de texto, etc.

- Interruption
  http://en.wikipedia.org/wiki/Real-time_operating_system
- RR scheduling
  http://en.wikipedia.org/wiki/Round-robin_scheduling
- Busy-waiting
  http://en.wikipedia.org/wiki/Busy_waiting
- Sugerencias de otra universidad
  http://www.ida.liu.se/~TDDB68/labs/lab1.shtml

Reloj de alarma

Estructura de Datos

A1: Copiar aquí la declaración de cada struct nueva o modificada, variable global o estática, "typedef", o enumeración. En 30 palabras o menos, identifique o propósito.

int64_t wake; Guarda la cantidad de tick en los cuales el hilos esta durmiendo.

struct list donorlist; Esta es una lista ascendiente de threads que están durmiendo.

Algoritmos

A2: Brevemente describa que sucede en una llamada a timer_sleep(), incluyendo los efectos del manejador de interrupciones del timer.

En las llamadas a timer_sleep():

- El sleep_ticks del hilo actual se le asigna los ticks que se dormirá mas los ticks actuales
- Apagar las interrupciones.
- El hilo se inserta en la lista de donantes
- Bloqueo el hilo
- Restablecer el nivel de interrupciones.

Entonces, en el timer interruption handler,

1. Chequea la lista para ver si uno de los threads necesita ser despertado
2. Si hay alguno, reasigna el sleep_ticks del hilo
3. Apaga las interrupciones
4. Remuevelo de la lista de dormidos
5. desbloquea el hilo
6. Restablecer el nivel de interrupciones.

SINCRONIZACION

A4: ¿Cómo se evitan las condiciones de competencia cuando múltiples hilos llaman a timer_sleep() simultáneamente?

Las listas de operaciones que pasan durante las interrupciones son desactivadas.

Decisiones de diseño

A6:¿Por que se eligió este diseño? ¿ En que formas es superior a otro diseño considerado?

Tener una lista y guardar los hilos durmiendo es algo que se piensa de una vez y

PLANIFICACION DE PRIORIDAD

ESTRUCTURA DE DATOS

B1: Copiar aquí la declaración de cada struct nueva modificada, variable global o estática, typedef, o enumeración. En 30 palabras o menos, identifique su propósito.

Se a;adio un struct thread

/* Mantener el control de los hilos en base a su prioridad ante de la donación*/

int priority_original;

/*Si la prioridad de un hilos es donada */

short is_donated;

/* Una lista en orden descendente por los hilos de mayor valor de prioridad esperando por el semáforo, representa todo los bloqueos que un hilos tiene, usado en múltiples donaciones para asignar la prioridad de un hilo dado la cantidad de bloqueos que tiene */

struct list locks;

/*Hilo bloqueado por el bloqueo, usa una donación anidada para saber quien va después */

struct lock *lock_blocked_by;

B2: Explique la estructura de datos utilizada para dar seguimiento a la donación de prioridad. Utilice arte ASCII para diagramar la donación anidada.

Como es mencionado en la pregunta, se fue agregado priority_original, is_donated, locks, lock_blocked_by a thread, y a lock se le agrego elem_lock y priority_lock, para ayudar a la donación de prioridad.

Cada vez que un bloqueo es adquirido por un hilo, el bloqueo va a ser insertado en el bloqueo del hilo, el cual es un lista ordenada por la priority_lock en la estructura lock. Al mismo tiempo, cuando es liberada, se saca de la lista de dueños y aqui es donde elem_lock dentro de la estructura lock juega un papel.

En una sola donación, cuando el bloqueo esta siendo adquirido, la prioridad del lock_holder es chequeada, si es menor que el que esta adquiriendo el lock, la donación pasara. En esta implementación, la prioridad original del hilo va a ser cambiada. Luego is_donated se le asigna 1, si no lo era antes.

Luego se chequea el hilo esta bloqueado por otro lock, el cual es necesario para donaciones anidadas. Si es así, otra donación va a pasar el mismo procedimiento que fue mencionado excepto que el nuevo donante es el actual donador, el nuevo donador es el que tiene el bloqueo que bloquea al actual donador. Esto pasara hasta que no halla donadores.

Cuando un bloqueo es liberado, el bloqueo va a ser removido de la lista de bloqueos del hilo y luego viene el chequeo de que si múltiples donaciones pasaron a ese hilo anteriormente. Si la lista de bloqueos esta vacía, significa que no pasaron múltiples donaciones. En cambio, coge el primero bloque de la lista y si la prioridad original de esta no ha sido cambiada, el hilo renuncia a esta prioridad también. Dado que es los bloqueos están ordenado de manera descendente en la lista, se puede garantizar que el primero bloqueo de la lista va a tener la máxima prioridad, la cual es la que el dueño debería tener.

Usando la estructura de dato y el algoritmo mencionado, donación por prioridad, incluyendo una simple donación, una donación múltiple y una donación anidad puede ser alcanzada.

Toma esto por ejemplo
A hilo, prioridad 31, tiene el bloqueo lock_1.
B hilo, prioridad 32, tiene el bloqueo lock_2 y adquiere el bloqueo lock_1
C hilo, prioridad 33, adquiere el bloqueo lock_2

```
Paso 1: Al comienzo:
=========================
.-------------------------------------------------.
|                Hilo A (Comenzando)              |
+-------------------+-----------------------------+
| Miembro           | valor                       |
+-------------------+-----------------------------+
| priority          |                          31 |
| priority_original |                          31 |
| is_donated        | false                       |
| locks             | {lock_1 (priority_lock = -1)} |
| lock_blocked_by   | NULL                        |
'-------------------+-----------------------------'
.-------------------------------------------------.
|                Hilo B (Comanzando)              |
+-------------------+-----------------------------+
| Miembro            | valor                      |
+-------------------+-----------------------------+
| priority          |                          32 |
| priority_original |                          32 |
```

```
| is_donated        | false                          |
| locks             | {lock_2 (priority_lock = -1)} |
| lock_blocked_by   | NULL                           |
'-------------------+--------------------------------'
.--------------------------.
|    Hilo C (Comenzando)    |
+-------------------+-------+
| Miembro            | valor|
+-------------------+-------+
| priority          |    33 |
| priority_original |    33 |
| is_donated        | false |
| locks             | {}    |
| lock_blocked_by   | NULL  |
'-------------------+-------'
================================================================

Paso 2: B adquiere lock_1:
==========================
.-------------------------------------------------.
|              Hilo A (B adquiere L1)             |
+-------------------+-----------------------------+
| Miembro            | valor                      |
+-------------------+-----------------------------+
| priority          |                         31 |
| priority_original |                         32 |
| is_donated        | true                       |
| locks             | {lock_1 (priority_lock = 32)} |
| lock_blocked_by   | NULL                       |
'-------------------+-----------------------------'
.-------------------------------------------------.
|              Hilo B (B adquiere L1)             |
+-------------------+-----------------------------+
| Miembro            | valor                      |
+-------------------+-----------------------------+
| priority          |                         32 |
| priority_original |                         32 |
| is_donated        | false                      |
| locks             | {lock_2 (priority_lock = -1)} |
| lock_blocked_by   | &lock1                     |
'-------------------+-----------------------------'
.--------------------------.
|  Hilo C (B adquiere L1)   |
+-------------------+-------+
| Miembro            | valor|
+-------------------+-------+
| priority          |    33 |
| priority_original |    33 |
| is_donated        | false |
| locks             | {}    |
| lock_blocked_by   | NULL  |
'-------------------+-------'
================================================================

Paso 3-1: C adquiere lock_2:
============================
.-------------------------------------------------.
|          Hilo B (C adquiere L2, Paso 1)         |
```

```
+------------------+------------------------------+
| Miembro           | valor                        |
+------------------+------------------------------+
| priority          |                           32 |
| priority_original |                           33 |
| is_donated        | true                         |
| locks             | {lock_2 (priority_lock = 33)} |
| lock_blocked_by   | &lock1                       |
'------------------+------------------------------'
.---------------------------------.
| Hilo C (C adquiere L2, Paso 1)   |
+---------------------+-----------+
| Miembro              | valor    |
+---------------------+-----------+
| priority            |        33 |
| priority_original   |        33 |
| is_donated          | false     |
| locks               | {}        |
| lock_blocked_by     | &lock_2   |
'---------------------+-----------'
.-------------------------------------------------.
|          Hilo A (C adquiere L2, Paso 1)          |
+------------------+------------------------------+
| Miembro           | valor                        |
+------------------+------------------------------+
| priority          |                           31 |
| priority_original |                           32 |
| is_donated        | true                         |
| locks             | {lock_1 (priority_lock = 32)} |
| lock_blocked_by   | NULL                         |
'------------------+------------------------------'
=====================================================================

Paso 3-2: C adquiere lock_2:
============================
.-------------------------------------------------.
|          Hilo B (C adquiere L2, Paso 2)          |
+------------------+------------------------------+
| Miembro           | valor                        |
+------------------+------------------------------+
| priority          |                           32 |
| priority_original |                           33 |
| is_donated        | true                         |
| locks             | {lock_2 (priority_lock = 33)} |
| lock_blocked_by   | &lock1                       |
'------------------+------------------------------'
.---------------------------------.
| Hilo C (C adquiere L2, Paso 2)   |
+---------------------+-----------+
| Miembro              | valor    |
+---------------------+-----------+
| priority            |        33 |
| priority_original   |        33 |
| is_donated          | false     |
| locks               | {}        |
| lock_blocked_by     | &lock_2   |
'---------------------+-----------'
.-------------------------------------------------.
```

```
|          Hilo A (C adquiere L2, Paso 2)           |
+------------------+------------------------------+
| Miembro           | valor                        |
+------------------+------------------------------+
| priority          |                           31 |
| priority_original |                           33 |
| is_donated        | true                         |
| locks             | {lock_1 (priority_lock = 32)} |
| lock_blocked_by   | NULL                         |
'------------------+------------------------------'
================================================================

Paso 4: A suelta lock_1:
=========================
.-----------------------------.
| Hilo A (A suelta lock_1))     |
+--------------------+---------+
| Miembro             | valor  |
+--------------------+---------+
| priority           |      31 |
| priority_original  |      31 |
| is_donated         | false   |
| locks              | {}      |
| lock_blocked_by    | NULL    |
'--------------------+---------'
.-------------------------------------------------.
|            Hilo B (A suelta lock_1)               |
+------------------+------------------------------+
| Miembro           | valor                        |
+------------------+------------------------------+
| priority          |                           32 |
| priority_original |                           33 |
| is_donated        | true                         |
| locks             | {&lock_2 (priority_lock = 33), |
|                   |  &lock_1 (priority_lock = 32)} |
| lock_blocked_by   | NULL                         |
'------------------+------------------------------'
.-----------------------------.
| Hilo C (A suelta lock_1)     |
+--------------------+---------+
| Miembro             | valor  |
+--------------------+---------+
| priority           |      33 |
| priority_original  |      33 |
| is_donated         | false   |
| locks              | {}      |
| lock_blocked_by    | &lock_2 |
'--------------------+---------'
================================================================

Paso 5: B suelta lock_2:
=========================
.-----------------------------.
| Hilo A (B suelta lock_2)) |
+--------------------+---------+
| Miembro             | valor  |
+--------------------+---------+
| priority           |      31 |
```

```
| priority_original   |      31 |
| is_donated          | false   |
| locks               | {}      |
| lock_blocked_by     | NULL    |
'---------------------+---------'
.-------------------------------------------------.
|            Hilo B (B suelta lock_2)               |
+-------------------+-----------------------------+
| Miembro            | valor                        |
+-------------------+-----------------------------+
| priority          |                           32 |
| priority_original |                           32 |
| is_donated        | false                        |
| locks             | {&lock_1 (priority_lock = 32)} |
| lock_blocked_by   | NULL                         |
'-------------------+-----------------------------'
.-------------------------------------------------.
|            Hilo C (B suelta lock_2)            |
+-------------------+-----------------------------+
| Miembro            | valor                        |
+-------------------+-----------------------------+
| priority          |                           33 |
| priority_original |                           33 |
| is_donated        | false                        |
| locks             | {&lock_2 (priority_lock = 33)} |
| lock_blocked_by   | NULL                         |
'-------------------+-----------------------------'
=====================================================================
```

ALGORITMOS

B3: ¿ Como se asegura que el hilo de mayor prioridad esperando un lock, semáforo, o variable condicional sea el que se despierte primero?

A: Cambiar la lista de espera por una lista sorteada, la cual esta ordenada por prioridad. Cada vez que se levante uno que estaba esperando, los hilos son puesto en la lista de listos desde el inicio al final, en el cual estarán desde la prioridad mas alta a la baja.

B4: Describir la secuencia de eventos cuando una llamada a lock_acquire() causa una donación de prioridad. ¿Como se maneja esta donación anidada?

Pasos:

1. Desactivar los interruptores
2. Donación
    1. Si lock_holder es NULL
        1. sema_down: si el valor de sema es 0, poner a todo los hilos adquieran este bloqueo en la lista de espera hasta que el valor de sema se vuelva positivo
        2. Asignar el hilo actual a obtener el bloqueo
            2. Sino compara la prioridad de lock_holder (L) con la prioridad del hilo actual (A):

1. Si la prioridad de L > la prioridad de C
    1. Has sema_down hasta que el valor de sema se vuelva positivo que significa que

el lock fue soltado
    2. Asignar el hilo actual como el lock_holder
    2. Sino
        1. [Donacion] Asigna la prioridad de L a C
        2. Has sema_down, hasta que el lock sea soltado.
        3. El hilo actual se convierte en el lock_holder
2. Restablece las interrupciones en el estado que estaban antes de ser deshabilitadas.

B5: Describir la secuencia de eventos cuando lock_release() es llamado en un lock que un hilo mayor prioridad esta preparado.

A: Pasos:

1. Asegurar que este hilo es el dueño del bloqueo. Si no lo es reportar que hay un error
2. Desactivar las interrupciones
3. Asignar el lock_holder a NULL
4. Hacer sema_up: incrementar el valor de sema por 1, lo cual significa que el lock puede ser obtenido por el semáforo de los esperando o cualquier hilo va a obtenerlo.
5. Asignar el valor de prioridad al dueño original del bloqueo
    1. Si ninguna donación pasa, asignar el valor que tenia anteriormente
    2. Sino
        1. Si el lock_holder original solo tiene este bloqueo
            1. Asignar el valor que tenia anteriormente
        2. Sino (donaciones anidadas)
            1. Asignar la máxima prioridad en su lista de bloqueo al valor de la prioridad del lock_holder

Después de que este bloqueo es soltado, el valor del sema va a incrementar por 1 y el valor de sema sera positivo. El hilos con mayor prioridad que esta esperando obtendrá este bloqueo.

SINCRONIZACION

B6: Describir una condición de carrera potencial en thread_set_priority() y explicar como su implementación la evita. ¿Se puede utilizar un lock para evitar esta carrera?

Durante la donación de prioridad, la prioridad del lock_holder sera la de su donante, al mismo tiempo puede que el hilo mismo quiera cambia su prioridad. Si el donante y el mismo hilo cambian su prioridad en diferente orden, pueden causar un resultado diferente.

Se desactivan la interrupciones para prevenir que esto pase. En esta implementación, ya que no fue proporcionado una interfaz y estructura para compartir un lock entre el donante y el mismo hilo.

DECISIONES DE DISEÑO

B7: ¿Por que eligieron este diseño? ¿ En que forma es superior a otro diseño considerado?

Al comienzo, para evitar crear una estructura thread y lock grande, no se agrego is_donated en la estructura thread, ni priority_lock en la estructura lock.

En vez de usar is_donated, se utilizo los cambios de la prioridad origina para indicar si la prioridad de un hilo es donada. Por ejemplo, se inicializaba la prioridad a -1 cuando era creada. Cada vez que una donación pasaba, se le asignaba el valor de prioridad del thread justo ante de ser hecha la donación. Se comparaba los valores de prioridad con la original, si eran diferente y la prioridad original ha sido cambiada por otro valor, entonces esto significa que una donación paso. La prioridad original tiene dos funciones: 1 guardar el valor de la prioridad justo antes de la donación y la segunda indicar si paso o no una donación. Habrá un problema cuando una prioridad de menor valor esta envuelta. Si el nuevo valor de la prioridad es mayor que el actual, ambos van a cambiar a la nueva prioridad. Esto sigue siendo el proceso de donación, pero si sus valores son el mismo, entonces la donación no pasara. Pero no es verdad. Así que por esta razón se agrego is_donated para saber si una donación ocurrió.

Para la estructura lock, en vez de agregar un nuevo miembro en lock, se puede conseguir el de mayor prioridad en la lista de espera haciendo esto:

```
list_entry (list_front (lock->semaphore.waiters), struct thread, elem)->priority.
```

Pero el código se volvería mucho mas complejo.

Para las múltiples donaciones, después de que un hilos libera un bloqueo, su prioridad debe ser la mayor prioridad en la lista de espera. Para poder conseguir múltiples donaciones, se necesita siempre tener en cuenta la mayor prioridad de ese bloqueo. En el comienzo, en vez de utilizar una lista de bloqueo dentro de la estructura thread, se pensó que se podia mantener el de mayor prioridad en la lista de espera. Pero esto no funcionara ya que cuando otro hilo adquiera este bloqueo, se tiene que reemplazar el hilo

PLANIFICADOR AVANZADO

ESTRUCTURAS DE DATOS

C1: Copie aquí las declaraciones de cada nueva o modificada “struct”, variable global o estática, “typedef”, o enumeración. Identifique su propósito en 25 palabras o menos.

struct thread

int nice; /* hilo buen valor */

int recent_cpu /*hilo CPU reciente */

En thread.h

/* Limite de buen valor */

#define NICE_MIN -20

#define NICE_DEFAULT 0

#define NICE_MAX 20

ALGORITMOS

C2: Suponga que los hilos A, B y C tienen valores nice de 0, 1 y 2 respectivamente. Cada uno tiene un valor ‘recent cpu‘ de 0. Llene la siguiente tabla mostrando la decisión de planificación, la prioridad y valor ‘recent cpu‘ para cada hilo luego de los ticks de reloj dados:

| recent_cpu | priority |

timer ticks	A	B	C	A	B	C	hilo para correr	Nota
0	0	1	2	63	61	59	A	
4	4	1	2	62	61	59	A	
8	7	2	4	61	61	58	B	Round Robin
12	6	6	6	61	59	58	A	
16	9	6	7	60	59	57	A	
20	12	6	8	60	59	57	A	
24	15	6	9	59	59	57	B	Round Robin
28	14	10	10	59	58	57	A	
32	16	10	11	58	58	56	B	Round Robin
36	15	14	12	59	57	56	A	

C3: ¿Alguna ambigüedad en la especificación del planificador hace que los valores en la tablas sean inciertos? Si es así, ¿qué regla utilizó para resolver estas ambigüedades? ¿Esto es compatible con el comportamiento de su planificador?

recent_cpu es ambiguo aqui. Cuando se calcula recent_cpu, no se fue considerado el tiempo del CPU gasta en las operaciones cada 4 ticks, como load_avg, recent_cpu para todos los hilos, priority para todo los hilo en todas las listas. Cuando CPU hace estas operaciones, el hilo actual necesita parar. Entonces cada 4 ticks, el tick real que es a;adido a recent_cpu (recent_cpu incrementa 1 en cada tick) no es en realidad 4 ticks. No se pudo calcular cuanto tiempo fue gastado. Asi que lo que se le hizo fue incrementar 4 ticks al recent_cpu cada 4 ticks.

En la implementación del planificador se realizo lo mismo que aqui.

C4: Considere la manera en la que divide el costo de planificación entre código dentro y fuera del contexto de interrupción. ¿Qué tan probable es que esta afecte el rendimiento?

Si el CPU gasta demasiado tiempo en las operaciones de recent_cpu, load_avg y priority, entonces tomara la mayor parte del tiempo al hilos actual . Así que este hilo no tendrá mucho tiempo de corrida como es esperado. Esto podría causar que el mismo incremente su load-avg, recent_cpu y disminuya su prioridad. Esto puede afectar a las decisiones del planificador.

JUSTIFICACIÓN

C5: Critique brevemente su diseño, apuntando ventajas y desventajas en sus decisiones. Si tuviera tiempo extra para trabajar en esta parte del proyecto, ¿cómo elegiría refinar o mejorar su diseño?

En el diseño no se aplico las 54 colas. Solo se utilizo una, la ready_list que Pintos tenia originalmente. Pero esta estaba en orden descendente al inicio, que cada vez que se insertaba un hilo a la ready_list, se insertara en orden. El tiempo de complejidad is O(n). Cada 4 ticks, era necesario calcular la prioridad de todo los hilos en la lista. Después de esto, se ordenaba la ready_list lo que tomaba $O(nlog(n))$. Ya que se necesitaba hacer esto cada 4 ticks, podría hacer que el hilo corriera menos de lo esperado. Si el *n* se volvía grande, el cambio de hilos pasaría con

bastante frecuencia. Si se hubiera usado 64 colas para los hilos que estén listos, donde estuvieran almacenados como un arreglo de colas y el indice indicaría el valor de prioridad. Cuando se necesitara agregar un thread se haría en $O(1)$. Después de los cálculos de prioridad cada 4 ticks, se tomaría $O(n)$ para volver a insertar cada thread.