Auxiliar 14: Repaso Examen

Examen 2017 - 2

CC4302 - Sistemas Operativos

José Astorga

Spinlocks - Herramientas de Sincronización de bajo nivel

29S 4O

Pregunta 1:

(i) ¿Tiene sentido usar spin-locks en una máquina monocore? Explique. Responda entonces cómo implementaría la exclusión mutua en un núcleo moderno de Unix para una máquina monocore.

Pregunta 1:

(i) ¿Tiene sentido usar spin-locks en una máquina monocore? Explique. Responda entonces cómo implementaría la exclusión mutua en un núcleo moderno de Unix para una máquina monocore.

Solución: No, no tiene sentido porque si un thread A intenta cerrar un spin-lock pero ese spin-lock lo posee otro thread B, el thread A hará busy-waiting inútilmente ocupando el 100% del tiempo del único core disponible. El thread A impedirá que el thread B se ejecute hasta que se acabe la tajada de tiempo del thread A. Recién ahí el thread B podrá liberar el spin-lock.

En vez de cerrar y abrir un spin-lock se deben inhibir las interrupciones antes de entrar a la sección crítica y activarlas nuevamente al salir. Así se asegura que un thread ejecutará toda la sección crítica de principio a fin, excluyendo la posibilidad de que una interrupción gatille un cambio de contexto que le dé la oportunidad a otro thread de ejecutarse en el único core disponible.

Spinlocks - Herramientas de Sincronización de bajo nivel

29S 4O

Pregunta 1:

(ii) Ud. debe elegir una herramienta para garantizar la exclusión mutua en una sección crítica de un módulo del núcleo de Linux. Considere una máquina multicore. ¿Bajo qué condiciones sería más eficiente usar un spin-lock y cuando sería más eficiente un semáforo?

Spinlocks - Herramientas de Sincronización de bajo nivel

29S 4O

Pregunta 1:

(ii) Ud. debe elegir una herramienta para garantizar la exclusión mutua en una sección crítica de un módulo del núcleo de Linux. Considere una máquina multicore. ¿Bajo qué condiciones sería más eficiente usar un spin-lock y cuando sería más eficiente un semáforo?

Solución: Depende de cuanto tiempo de CPU requiera la sección crítica. Si requiere muchos microsegundos conviene usar un semáforo porque si ya hay otro proceso en la sección crítica, se hará un cambio de contexto para retomar otro proceso, y así aprovechar mejor la CPU. En cambio, si la sección crítica es de corta duración, es mejor usar un spin-lock que hará busy-waiting para esperar que el otro core salga de la sección crítica. Si se usara un semáforo, el cambio de contexto tomaría más tiempo de CPU que el tiempo que toma la ejecución de la sección crítica.

Paginamiento
11O 13O 18O Aux9

Pregunta 1:

(iii) ¿Qué es lo más importante que perderían los usuarios si los computadores modernos no ofrecieran paginamiento (ni segmentación)? Explique considerando qué pasa cuando se “caen” aplicaciones como chrome, word, windows media player, waze, whatsapp, etc. debido a problemas con los punteros.

Paginamiento
11O 13O 18O Aux9

Pregunta 1:

(iii) ¿Qué es lo más importante que perderían los usuarios si los computadores modernos no ofrecieran paginamiento (ni segmentación)? Explique considerando qué pasa cuando se “caen” aplicaciones como chrome, word, windows media player, waze, whatsapp, etc. debido a problemas con los punteros.

Solución: Se perdería la protección entre procesos. Si una aplicación se “cae” por mal manejo de punteros, podría modificar la memoria de otra aplicación haciendo que esta también se “caiga”. Por lo tanto, cada vez que una sola aplicación se “cae” habría que reiniciar el sistema operativo completo y todas las aplicaciones. Con paginamiento basta reiniciar la aplicación que se “cayó”.

SSD vs HDD
17N

Pregunta 1:

(iv) Compare ventajas y desventajas de usar un disco tradicional para paginamiento versus un solid state drive (SSD). En su comparación considere número de page faults atendidos por segundo, velocidad (en MB/seg.) para cargar en memoria un proceso que fue llevado a disco (estado swap) y tiempo de vida del dispositivo de paginamiento.

Solución:

	Disco	SSD
page faults		
velocidad		
tiempos de vida		

SSD vs HDD
17N

Pregunta 1:

(iv) Compare ventajas y desventajas de usar un disco tradicional para paginamiento versus un solid state drive (SSD). En su comparación considere número de page faults atendidos por segundo, velocidad (en MB/seg.) para cargar en memoria un proceso que fue llevado a disco (estado swap) y tiempo de vida del dispositivo de paginamiento.

Solución:

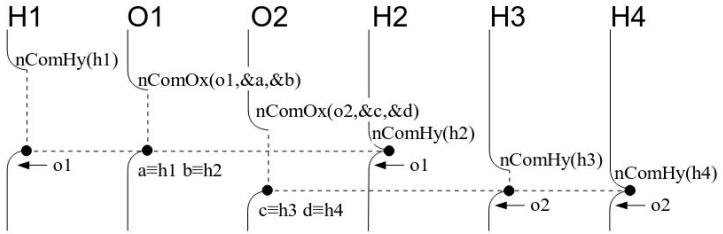
	Disco	SSD
page faults	~100 por segundo	~40000 por segundo
velocidad	100 MB por segundo	400 MB por segundo
tiempos de vida	5 años	1000 a 10000 escrituras por celda

Implementación de Herramientas de Sincronización
Aux7 20S 22S 27S

Pregunta 2

a.- (4,5 puntos) Se requiere implementar de manera nativa en nSystem las siguientes funciones:

```
void nCombineOxy(Oxygen *oxy, Hydrogen **pH1, Hydrogen **pH2);
Oxygen *nCombineHydro(Hydrogen *hydro);
```



La figura de arriba muestra la sincronización requerida. La tarea **H1** aporta un átomo de hidrógeno h1 llamando a `nCombineHydro`. Debe esperar por otro átomo de hidrógeno y uno de oxígeno. La tarea **O1** aporta el átomo de oxígeno o1 invocando `nCombineOxy`. Finalmente la tarea H2 aporta el último átomo de hidrógeno h2. Entonces recién **H1**, **O1** y **H2** pueden continuar. **H1** y **H2** retornan o1 y `nCombineOxy` entrega h1 en a y h2 en b. La misma sincronización ocurre con las tareas **O2**, **H3** y **H4**.

Implementación de Herramientas de Sincronización
Aux7 20S 22S 27S

Pregunta 2

Implemente las funciones `nCombineOxy` y `nCombineHydro` usando los procedimientos de bajo nivel de nSystem (`START_CRITICAL, schedule, nth_putBack, etc.`). Ud. no puede usar otros mecanismos de sincronización ya disponibles en nSystem como semáforos, monitores, mensajes, etc. Declare las variables globales que necesite y especifique los campos que agrega al descriptor de tarea. La ocupación de los átomos debe hacerse por orden de llegada. Debe evitar cambios de contexto innecesarios. Use una cola de nThreads (NthQueue) para las tareas que aportan hidrógeno y otra cola para las tareas que aportan oxígeno. Considere que la función `nth_queueLength(q)` entrega el largo de la cola q.

Pregunta 2

```
// Añadir campos oxy e hydro para el descriptor del thread
typedef struct {
    ...
    Oxygen oxy;
    Hydrogen hydro;
} *nThread;

// Añadir nuevos estados para los threads
typedef enum {
    ...
    WAIT_HYDRO,
    WAIT_OXY
} State;
```

Pregunta 2

```
NthQueue hydroQ; // = MakeQueue()
NthQueue oxyQ; // = MakeQueue()

void nOxygen(Oxygen *oxy, Hydrogen **pH1, Hydrogen
**pH2) {
// Garantizar exclusión mutua con S_C y E_C en
ambas funciones
   START_CRITICAL
   nThread thisTh= nSelf();

   if (nth_queueLength(hydroQ) < 2) {
       nth_putBack(oxyQ, thisTh);
       // Suspender tarea si no están los 2
hidrógenos
       suspend(WAIT_HYDRO);
       schedule();
   }
```

```
   nThread t1 = nth_getFront(hydroQ); // Extraer 2
tareas hidrógeno
   nThread t2 = nth_getFront(hydroQ);
   t1-> oxy = t2->oxy = oxy; // Depositar oxígeno
en tareas hidrógeno

   *pH1= t1->hydro; // Entregar hidrógenos en *pH1
y *pH2
   *pH2= t2->hydro;

   // Retomar las 2 tareas hidrógeno (no es
relevante el orden en que se retoman)
   setReady(t1);
   setReady(t2);
   schedule();

   END_CRITICAL
}
```

```
nOxygen *nHydrogen(Hydrogen *h) {
   START_CRITICAL
   nThread thisTh = nSelf();
   // Registrar hidrógeno para ser extraído por la
tarea oxígeno
   thisTh->hydro= h;
   // Colocar tarea en hydroQ para suspenderla en
(*)
   nth_putBack(hydroQ, thisTh);
   // Retomar una tarea oxígeno si hay 2 hidrógenos
en hydroQ
```

```
  if (nth_queueLength(hydroQ) >= 2 &&
       !nth_emptyQueue(oxyQ))
   {
       nTask t= nth_getFront(oxyQ);
       setReady(t);
   }
   suspend(WAIT_OXY);
   schedule(); // (*) Suspende tarea hidrógeno
   Oxygen *oxy= thisTh -> oxy; // Rescatar oxígeno
y retornarlo

   END_CRITICAL
   return oxy;
}
```

Scheduler del Disco
17N

Pregunta 2

b.- (1,5 puntos) 5 procesos se encuentran en estado de espera porque hicieron peticiones para leer bloques del disco en el siguiente orden: 300, 600, 200, 900, 800. El último bloque leído fue el 500 y el penúltimo el 700. Indique en qué orden se harán las lecturas de estos 5 procesos cuando la estrategia de scheduling de disco es: (i) first come first served, (ii) shortest seek first, (iii) método del ascensor (o look).

Scheduler del Disco
17N

Pregunta 2

b.- (1,5 puntos) 5 procesos se encuentran en estado de espera porque hicieron peticiones para leer bloques del disco en el siguiente orden: 300, 600, 200, 900, 800. El último bloque leído fue el 500 y el penúltimo el 700. Indique en qué orden se harán las lecturas de estos 5 procesos cuando la estrategia de scheduling de disco es: (i) first come first served, (ii) shortest seek first, (iii) método del ascensor (o look).

Sol:

(i) 300, 600, 200, 900, 800

(ii) 600, 800, 900, 300, 200

(iii) 300, 200, 600, 800, 900

Spinlocks - Herramientas de Sincronización de bajo nivel

29S 4O Aux8

Pregunta 3

I. (4,5 puntos) Considere una máquina multi-core en la que no existe un núcleo de sistema operativo y por lo tanto no hay un scheduler de procesos. La siguiente es una implementación incompleta del mismo problema de la pregunta 2 parte a.

```
Hydro *_hydro; // Buffer de 1 item
Oxy **_pOxy;
int _m= OPEN;  // mutex
// condición buffer vacio
int _empty= OPEN;
// condición buffer lleno
int _full= {CLOSED, CLOSED};
// 2 condiciones adicionales
int _wait[2]= CLOSED;
int _k= 0;
Oxygen *combineHydro(
  Hydrogen *hydro) {
  ... complete ...
  _k= (_k+1)%2;
  ... complete ...
}
```

```
void combineOxy(Oxy *oxy,
  Hydro **pH1, Hydro **pH2) {
  spinLock(&_m);
  spinLock(&_full); // wait buffer lleno
  *pH1= _hydro; // 1er átomo de hidrógeno
  *_pOxy= oxy;
  // signal buffer vacio
  spinUnlock(&_empty);
  spinLock(&_full); // wait buffer lleno
  *pH2= _hydro; // 2do átomo de hidrógeno
  *_pOxy= oxy;
  // signal 1er y 2do hidrógeno
  spinUnlock(&_wait[0]);
  spinUnlock(&_wait[1]);
  // signal buffer vacio
  spinUnlock(&_empty);
  spinUnlock(&_m);
}
```

En esta pregunta la ocupación de los átomos se hace en cualquier orden.Complete la función combineHydro sin alterar el resto de la implementación. Por claridad las variables globales empiezan con _. Observe que algunos spin-locks se usan como condiciones.

Spinlocks - Herramientas de Sincronización de bajo nivel

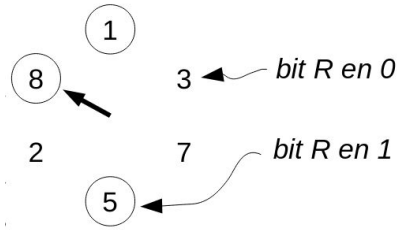
29S 4O Aux8

Pregunta 3

```
Oxygen combineHydro(Hydrogen *hydro) {
  spinLock(&_empty);          // wait buffer vacio
  _hydro= hydro;              // Colocar hidrógeno en buffer
  Oxy *oxy;
  _pOxy= &oxy;                // Dejar dirección de variable en donde debe quedar
                              // el oxígeno y retornar ese oxígeno (*)
  int slot= _k;
  _k= (_k+1)%2;
  spinUnlock(&_full);         // signal buffer lleno
  spinLock(&_wait[slot]);     // Esperar hasta completar molécula de agua:
  // En realidad no se puede usar &wait[k] debido a un datarace, pero se acepta.
  return oxy;                 // Junto con (*)
}
```

Paginamiento en Demanda - Estrategia del Reloj
11O 13O 18O Aux9

Pregunta 3



Algoritmo para estrategia del reloj:

```
while ( bitR( cursor( ) ) == 1 ) {
      setBitR( cursor( ) , 0 )
      avanzarCursor( )
}
reemplazarCursor( )
avanzarCursor( )
```

Paginamiento en Demanda - Estrategia del Reloj
11O 13O 18O Aux9

Pregunta 3

5, 7, 4, 7, 2, 6.