Clase Auxiliar IX

Prof: L. Mateu
Aux: M. Leyton

8 de octubre de 2004

1. SpinLocks (P1 C2 02/2002)

Un programador ha implementado un par de procedimientos que sirven para sincronizar *threads* que corren en modo sistema, dentro de un núcleo *multi-threaded* para multiprocesadores. Varios *threads* pueden eserar hasta que ocurra un cierto evento invocando el procedimiento `kWait()`. Un único *thread* notifica la ocurrencia del evento invocando el procedimiento `kNotify()`. Si un trhead invoca `kWait()` y el evento ya ocurrió, el thread continúa de inmediat (sin esperar). El programador implementó estos procedimientos usando spin-locks, de la siguiente manera:

```
/* spin-lock cerrado */
int event_spinLock=CLOSED
void kWait() {
    spinLock(&event_spinLock);
    spinUnlock(&event_spinLock);
}
void kNotify() {
/* abrir el spin-lock */
spinUnlock(&event_spinLock);
}
```

- Parte a.- Discuta si esta solución es correctad esde un punto de vista funcional (es decir un thread unca va a retornar de kWait antes de la invocación de kNotify y ningún thread se quedará esperando indefinidamente después de lainvocación de kNotify).

- Parte b.- Critique esa solución desde un punto de vista de la eficiencia. Considere que la notificación de evento puede ocurrr segundos o minutos después de la inovación de kWait

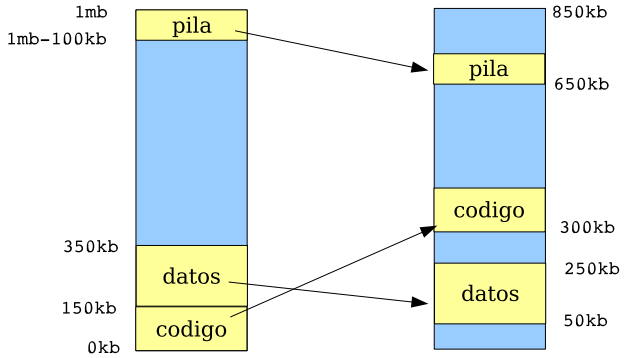
- Parte c.- Reprograme eficientemente ambos procedimientos utilizando los siguientes procedimientos de bajo nivel.

```
Queue kMakeQueue();
void kPutProc(Queue queue, kProc p);
void kPushProc(Queue queue, kProc p);
kProc kGetProc(Queue queue);
int kEmptyQueue(Queue queue);
```

```
void kResume();
Queue ready_queue;
int ready_queue_spinLock;
kProc currentProc();
```

2. Segmentación (P2 C2 02/2001)

La figura muestra a la izquierda el espacio de direcciones virtuales de u proceso Unx que ocupa 450KB de memoria. El proceso corre en un computador d 1MB con arquitectura segmentada. A la derecha se muestra la ubicación actual de los segmentos en la memoria real. Se dispone en total de 850KB para segments (el resto se ocupa en estructuras de datos del núcleo).



- Parte a. Suponga que el proceso invoca a sbrk para hacer crecer su área de datos de 200KB a 400KB. Modifique la figura para mostrar nua posible asignación de memoria después de la llamada a sbrk. Muestre en su figura las direcciones que ocupan ahora los segmentos en el espacio de direcciones virtuales y la memoria real.

- Parte b. Haga la tabla de segmentos del proceso después de invocar a sbrk. Indique base virtual, límite virtual, deplazamiento y atributos para cada segmento.

- Suponga que ahora el proceso invoca a fork. Explique como logra el núcleo del sistema operativo satisfacer este nuevo requerimiento a pesar que no hay memoria disponible para todo los segmentos del nuevo proceso.

3. Solución SpinLocks

- a) Efectivamente la solucion se encuentra correcta desde un punto de vista funcional. Mientras no se invoque kNotify(), todas las tareas que llamen a kWait() se quedaran bloqueadas. Una vez que ejecute kNotify(), una de las tareas que estaba bloqueada en kWait sera despertada y ejecutara el spinUnlock despertando a la siguiente tarea. Asi, una a una se continuaran despertando una las demas tareas.
- b) Como los spin-locks realizan busy-waiting, esta solucion es muy costosa en tiempo de CPU, ya que todas las tareas bloqueadas en spinLock continuan sentadas en la cola ready.

c)

```
int notify_flag = FALSE;
int even_spinLock= OPEN;
Queue kwaitQ=kMakeQueue();

void kWait(){
    spinLock(&event_spinLock);
    if(!notify_flag){
        currentProc()->status= WAIT_EVENT;
        kPutProc(kwaitQ, currentProc());
        spinUnlock(&event_spinLock);
        kResume();
    }
    else spinUnlock(&event_spinLock);
}

void kNotify(){
    spinLock(&event_spinLock);
      notify_flag=TRUE; /* cambiamos el flag */
      spinLock(&ready_queue_spinLock);

        /* colocamos las taras en la cola ready */
        while(!kEmptyQueue(kwaitQ)){
          kProc proc = kGetProc(kwaitQ);
          proc->status=READY;
          kPutProc(ready_queue,proc);
        }
        kPushProc(ready_queue, currentProc());

      spinUnlock(&ready_queue_spinLock);
    spinUnlock(&event_spinLock);
    kResume();
}
```

4. Solución Segmentación

- a) ...
- b)

inicio virutal	fin virutal	desplazamiento	atributos	correspondencia
1mb-100kb	1mb	650kb	rw	pila
150kb	550kb	50kb	rw	datos
0kb	150kb	500kb	r	código

- c) Porque puede mandar un proceso completo a disco.