

Clase Auxiliar XII

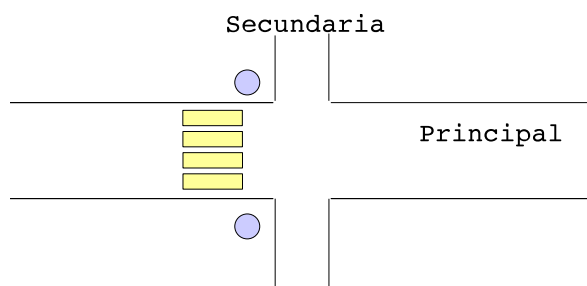
Prof: L. Mateu

Aux: M. Leyton

15 de noviembre de 2004

1. P1 Examen 97/02

Se desea implementar el software de control de un semáforo para el cruce de calles de la siguiente figura:



La luz verde de la calle principal dura 60 segundos (si no hay peatones que deseen cruzar) mientras que la luz verde de la calle secundaria dura sólo 30 segundos ya que tiene menor circulación. La luz amarilla de ambas calles dura 3 segundos. Por lo tanto la luz roja de la calle principal dura 33 segundos y la de la calle secundaria 63 segundos.

La calle principal tiene además un cruce peatonal. Los peatones disponen de un botón que pueden presionar para acelerar la luz verde que les permite cruzar. Cuando se ha presionado este botón, la luz verde de la calle principal se reduce a 30 segundos. Si el botón se presiona pasado los 30 segundos de luz verde de la calle principal, ésta se coloca en amarilla de inmediato.

Ya se han implementado los procedimientos que controlan las luces de ambas calles. Al invocar `luzPrimVerde()` la luz de la calle principal se coloca en verde. Este procedimiento retorna de inmediato, dejando la luz en verde hasta que se invoque `luzPrimAmarilla()` que la coloca en amarilla. Al invocar `luzPrimRoja()` la luz queda en rojo hasta que se invoque nuevamente `luzPrimVerde()`. Análogamente los procedimientos `luzSecVerde()`, `luzSecAmarilla()` y `luzSecRoja()` sirven para controlar el color de la luz de la calle secundaria.

Además se ha implementado el procedimiento `esperarBoton()` que bloquea la tarea que lo invoca hasta que un peatón presione el botón.

Implemente un procedimiento que controle las luces del cruce usando tareas de `nSystem`. Ud. puede lanzar tareas auxiliares si lo estima necesario.

2. P2 Examen 96

El siguiente procedimiento ordena un arreglo de n enteros usando bubble sort.

```
void bubblesort(int a[], int n){
    int i,j;
    for(i=n; i>1; i--){
        for(j=1; j<i; j++){
            if(a[j-1]>a[j]) swapint(&a[j-1], &a[j]);
        }
    }
}
```

Este procedimiento se ejecuta con un arreglo de 1MByte (256K palabras enteras) en un computador que posee memoria real para almacenar sólo la mitad.

1. ¿Podría correr este procedimiento si el computador posee una arquitectura segmentada. Explique
2. El procedimiento se ejecuta en una arquitectura que usa páginas de 4KBytes (o 1024 enteros) con un sistema operativo que implementa la estrategia de reemplazo LRU (least recently used). Haga una estimación del número de pagefaults que se producirá debido a páginas de datos.
3. Concluya sobre la utilidad de un sistema de memoria virtual en este caso.

3. P2 Examen 97 (parte c)

Ud. tiene un dilema. Utilizar un hash con linear-probing o hashing-doble. Los análisis matemáticos indican que con linear-probing habrá que visitar en promedio 2.5 filas por cada búsqueda, mientras que con hashing doble se necesitará visitar 1.8 filas por cada búsqueda.

Elija un esquema. Haga supuestos razonables.

Solucion P1

```
enum {BOTONAPRETADO, PRIMARIA, SECUNDARIA, OK, NOOK}

Task admin,buttonListener;
int currentverde;

void switchToSecundaria(){
    luzPrimAmarilla();
    nSleep(3);
    luzPrimRoja();
    luzSecVerde();
    currentverde=SECUNDARIA;
}

void switchToPrimaria(){
    luzSecAmarilla();
    nSleep(3);
    luzSecRoja();
    luzPrimVerde();
    currentverde=PRIMARIA;
}

void AdminPrim(){

    /* vaciamos si alguien apreto el boton
    * mientras estaba en roja la calle */
    while(nReceive(&sender,0))
        nReply(sender,NOOK);

    switchToPrimaria();
    int inicio=nGetTime();
    int nTask sender;
    int msg=(int)nReceive(&sender, 30);

    /* Dentro de los primero 30 segundos */
    if(sender!=NULL){
        nSleep(
            max(
                30-(nGetTime()-inicio),
                0)
        );

        nReply(sender, OK);
        return;
    }

    /* Esperamos 30 segundos o hasta que apreten el boton */
    msg=(int)nReceive(&sender, 30);
    if(sender!=NULL)
        nReply(sender,OK);
}

void AdminSec(){

    switchToSecundaria();
    nSleep(30);
}

void nMain(){

    buttonListener=nEmitTask(ButtonListener,NULL);

    while(true){
        AdminPrim();
        AdminSec();
    }
}

void ButtonListener(){

    while(true){
        esperarBoton();
        nSend(admin, BOTONAPRETADO);
    }
}
```

Solucion P2

1. No se puede, ya que con una arquitectura segmentada todo el segmento debe estar en memoria para correr, y en este caso el segmento de Datos o Pila tendria un tamaño mayor que la memoria real.
2. Considerando que solamente debe estar en memoria las paginas asoaciadas al arreglo, y despreciando paridad (par,impar):

$$n + (n - 1) + (n - 2) + \dots + \frac{n}{2}$$

$$\frac{(n+1)n}{2} - \frac{(k+1)k}{2}, \quad k = \frac{n}{2}$$

$$\frac{(n+1)n}{2} - \frac{(\frac{n}{2}+1)\frac{n}{2}}{2}$$

$$\frac{(\frac{3n}{2}+1)n}{4}$$

Solución P3

De acuerdo a los apuntes del curso los tiempos de acceso a memoria son de $t_m = 60 [ns]$ y $t_p = 12 [ms]$. Luego,

$$t_e = (1 - r) \cdot t_m + r \cdot t_p$$

Por lo que la diferencia entre un esquema y otro se da en la tasa r .

La idea es que en caso de colisiones, con linear probing, como sigues buscando secuencialmente, se cae en la misma pagina, disminuyendo asi los page-faults.

Con hashing doble como vuelves a calcular un indice completamente distinto, es seguro que vas a caer en otra pagina, aumentando los page-faults.

Por lo tanto, si hay penuria de memoria, linear probing se comporta mejor.

El supuesto aca es que se trata de un arreglo en donde los elementos contienen directamente la informacion. Esto se puede hacer en C. En Java los arreglos contienen punteros, por lo que aun cuando se visite secuencialmente el arreglo, los elementos visitados pueden estar repartidos en la memoria, aumentando los page-faults. En general los lenguajes orientados a objetos (basados fuertemente en punteros) tienden a producir mas page-faults.