

AESI

Prueba de evaluación continuada – nº 2

Viernes, 19 de Noviembre de 2004

Ejercicio 1

Dado un sistema informático que dispone del monitor *sar* instalado, se pide lo siguiente:

- ¿Cada cuánto tiempo se activa el monitor?

El monitor *sar* se ejecuta por defecto cada 5min. Este valor ha sido extraído del libro de la asignatura: “Evaluación y modelado del rendimiento de los sistemas informáticos” ya que no se ha podido comprobar en la práctica...

Además, leyendo el manual de *sar* (*man sar*) se puede apreciar que se puede llamar al ejecutable pasándole el intervalo de ejecución (en segundos) y el número de líneas que queremos apreciar:

```
# sar INTERVALO_EN_SEGUNDOS NÚMERO_DE_LINEAS
```

Por ejemplo:

```
# sar 2 5
Linux 2.6.9t41 (canela)          11/08/04

09:11:27      CPU      %user      %nice      %system      %iowait      %idle
09:11:29      all       4.46       0.00       1.98         0.00        93.56
09:11:31      all       1.01       0.00       0.50         0.00        98.49
09:11:33      all       0.50       0.00       0.50         0.00        99.00
09:11:35      all       0.00       0.00       0.00         0.00       100.00
09:11:37      all       0.50       0.00       0.99         0.00        98.51
Average:      all       1.30       0.00       0.80         0.00        97.90
```

- *Determinése el tamaño de un fichero histórico correspondiente a un día entero. Teniendo en cuenta el número de activaciones del monitor en un día, calcúlese el volumen que ocupa la información recogida en cada activación.*

Tal y como se puede leer en el *man* del *sar*, se puede ejecutar el monitor con la opción *-o* indicándole así que la información no la muestre por pantalla y que se guarde en un fichero binario:

```
# sar 2 5 -o /var/log/sysstat/sadd
```

Analizando varios de los ficheros de salida del *sar* y viendo sus tamaños, se puede deducir lo siguiente (resolviendo el sistema de dos ecuaciones obtenidas mediante dos pruebas):

1. Un fichero binario creado por el *sar* ocupa de base (sin ninguna línea) 1392bytes.
2. Cada línea añadida a dicho fichero ocupa 1152bytes.

Por tanto, y suponiendo que se invoca al monitor *sar* con un intervalo de X segundos, obtenemos el siguiente tamaño (en bytes):

$$\text{Tamaño} = 1392 + \left(1152 * \left(\frac{86400}{X}\right)\right) \text{ donde } 86400 \text{ son los segundos que tiene un día } (24 * 60 * 60).$$

Si tomamos como ejemplo, el valor por defecto definido en el libro (5min = 300seg) obtenemos que el tamaño del fichero sería 33178992bytes, es decir, unos 33MB, al cabo de un día de ejecución.

- *Planifíquese una sesión de medida de 5min que recoja información sobre la actividad del disco. El intervalo de medida será de 12seg.*

Ya que el enunciado no especifica que se pueda contar con el monitor *vmstat* se deberá realizar este apartado con el monitor *sar*. Este monitor cuenta con la opción *-b* en la cual se monitoriza y se muestran estadísticas de E/S y transferencias a disco.

Para realizar una medida que dure 5min (ó 300seg) con un intervalo de 12seg, se deberá invocar a *sar* de la siguiente manera:

```
# sar -b 12 25
```

Donde 25 es el número de líneas que se deben mostrar ($300 / 12 = 25$).

- *Compruébese la evolución de la actividad a lo largo de un día, con la ayuda de algún programa de visualización gráfica como gnuplot, referida a la utilización del procesador, el número de cambios de contexto y el uso del sistema de memoria.*

Para realizar esta prueba se ha invocado al monitor *sar* de la siguiente manera:

```
# sar 300 288 -o sar.log
```

Donde 288 es el número de líneas que se deben guardar (las correspondientes a un día de ejecución con un intervalo de ejecución de 5 minutos).

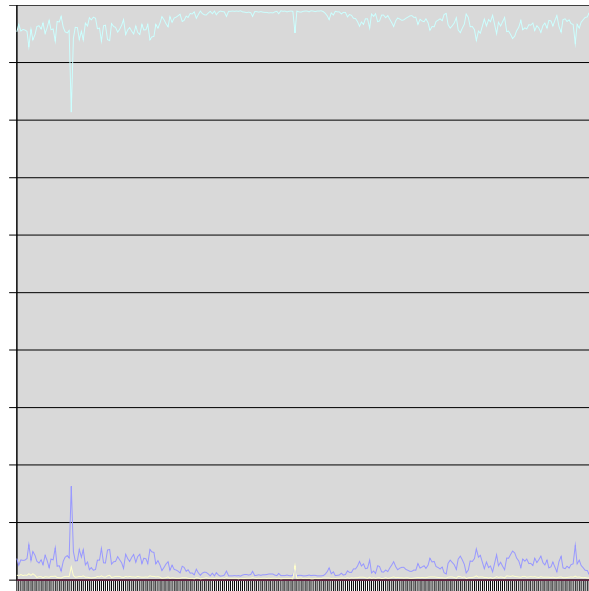
Para obtener los datos del uso del procesador y en formato CSV (fácil de importar mediante OOo) se ha hecho lo siguiente:

```
# sar -u -f sar.log -H > sar-u.txt
```

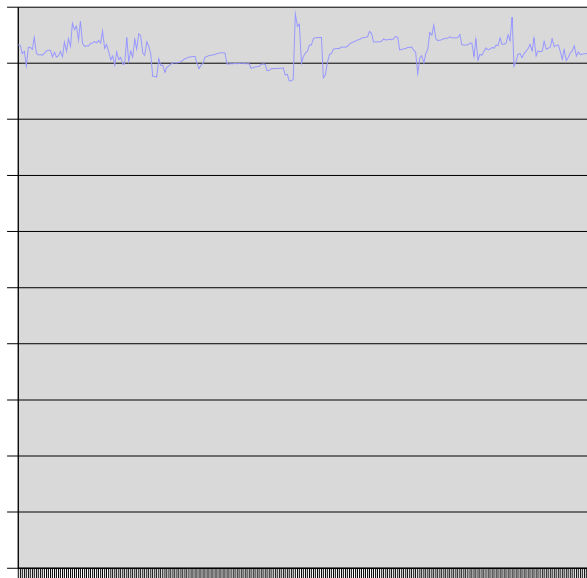
Realizaremos la misma operación para obtener el uso de memoria (y swap) y los cambios de contexto respectivamente:

```
# sar -r -f sar.log -H > sar-r.txt  
# sar -w -f sar.log -H > sar-w.txt
```

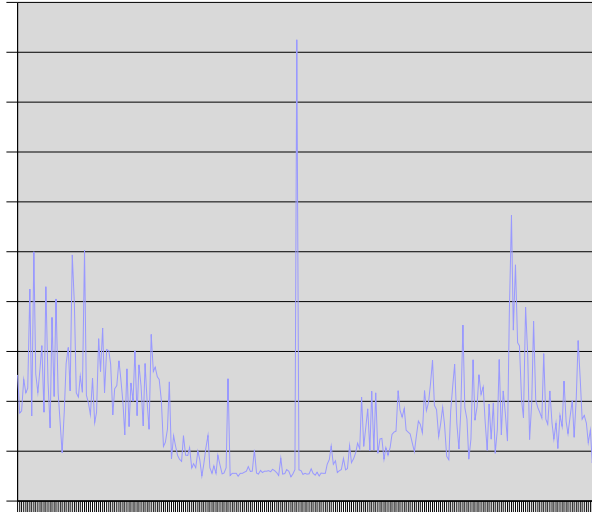
- Consumo de CPU:



- Consumo de memoria:



- Cambios de contexto:



Ejercicio 2

Considérese el programa siguiente que calcula, un número considerable de veces, el factorial de un número aleatorio entre 0 y 31. Este cálculo se efectúa de dos maneras diferentes. La primera de ellas mediante un algoritmo recursivo, y la segunda mediante un algoritmo iterativo.

```
#include <stdlib.h>
#define VECES 3000000

void main ()
{
    long a,b,i,j;

    for (i=1;i<=VECES;i++) {
        j=1+(long) (31.0*rand()/(RAND_MAX+1.0));
        a = factorial_recursivo(j);
        b = factorial_iterativo(j);
    }
}

long factorial_recursivo (long n)
{
    if (n == 0) return 1;
    return (n*factorial_recursivo(n-1));
}

long factorial_iterativo (long n)
{
    long i, fact = 1;

    for (i=1;i<=n;i++) fact = fact*i;
    return fact;
}
```

Respecto del programa anterior, se pide:

- Determina cuál de las versiones del algoritmo se ejecuta más rápidamente, y cuantificar la mejora en el tiempo de ejecución de la versión más rápida respecto de la más lenta.

Se ha compilado el programa de la siguiente manera:

```
gcc -o pac2-2 pac2-2.c -pg -g
```

Y se han seguido las instrucciones para monitorizar la ejecución del binario con *gprof* según las instrucciones encontradas en la página 42 del libro.

Viendo el fichero `pac2-2.gprof` resultante se ve que el tiempo total de ejecución es de 12,03 segundos. De los cuales el factorial recursivo ocupa el 48,71%, es decir, 5,86 segundos. El factorial iterativo ocupa el 31,38%, es decir, 3,78 segundos.

Por tanto, observando los tiempos de ejecución de ambos algoritmos con idénticos parámetros, podemos deducir que el algoritmo que calcula el factorial de manera iterativa es mucho más rápido que el algoritmo recursivo, concretamente un 55,02% más rápido.

- *Si un programa que tarda 45min en ejecutarse utiliza la función implementada con la versión recursiva del algoritmo durante el 72% del tiempo de ejecución. ¿Cuánto tiempo tardará si esta función se sustituye por la versión iterativa? ¿Qué mejora del rendimiento se producirá?*

El programa usa durante 32,4 min la versión recursiva del algoritmo y durante 12,6 min estará realizando otras tareas. Dado que la versión iterativa es un 55,02% más rápida, el algoritmo ahora tardará $32,4/1,5502 + 12,6 \text{ min} = 33,50 \text{ minutos}$.

Lo cual representa una mejora de aproximadamente el 47,88%.

Ejercicio 3

Un grupo de informáticos ha de diseñar una aplicación en lenguaje C que va a trabajar con matrices de grandes dimensiones de números reales. La aplicación realizará una serie de operaciones sobre estas matrices consistentes principalmente en el recorrido de sus elementos, para accesos de lectura y escritura. Con el objetivo de optimizar el diseño de la aplicación, el grupo de informáticos quiere averiguar, por un lado, si resulta más rápido acceder a los elementos de las matrices por filas o por columnas, y por otro, si es más costoso hacer una operación de escritura o de lectura en memoria. Se pide diseñar un programa que ponga de manifiesto, si las hay, estas diferencias, y ayude a cuantificarlas.

Nota: inténtese, en la medida de lo posible, reducir el efecto de la memoria caché del computador para que su existencia afecte lo menos posible a los resultados.

Para la resolución del problema se ha planteado el siguiente algoritmo para analizar mediante gprof (aplicando el mismo principio que en el ejercicio anterior).

```
#include <stdlib.h>

#define NVECES 3000
#define MAX1 347
#define MAX2 347

char m[MAX1][MAX2];

void escribe_filas (char c) {
    int i,j;
    for (i=0;i<MAX1;i++) {
        for (j=0;j<MAX2;j++) {
            m[i][j] = c;
        }
    }
}

void escribe_columnas (char c) {
    int i,j;
    for (j=0;j<MAX2;j++) {
        for (i=0;i<MAX1;i++) {
            m[i][j] = c;
        }
    }
}

void lee_filas (void)
{
    int i,j;
    char c;
    for (i=0;i<MAX1;i++) {
        for (j=0;j<MAX2;j++) {
            c = m[i][j];
        }
    }
}
```

```
    }  
}  
  
void lee_columnas (void)  
{  
    int i,j;  
    char c;  
    for (j=0;j<MAX2;j++) {  
        for (i=0;i<MAX1;i++) {  
            c = m[i][j];  
        }  
    }  
}  
  
int main (void)  
{  
    int i;  
    char c;  
    for (i=0;i<NVECES;i++) {  
        c = rand()%76 + 48;  
        escribe_filas(c);  
        c = rand()%76 + 48;  
        escribe_columnas(c);  
        lee_filas();  
        lee_columnas();  
    }  
    return 0;  
}
```

Analizando el fichero de salida del gprof se ve que el programa tarda en total 110,43 segundos en ejecutarse. Analizando cada función, comprobamos:

- escribe_columnas: 34,74% = 42,62 segundos
- escribe_filas: 28,50% = 33,53 segundos
- lee_columnas: 18,89% = 20,95 segundos
- lee_filas: 18,20% = 18,10 segundos

Analizando los tiempos de ejecución se ve claramente como un recorrido por filas, tanto en lectura como escritura, es mucho mas eficiente que un recorrido por columnas. La razón es sencilla si pensamos en como está estructurada la matriz en memoria ya que si recorremos por filas, iremos trabajando con posiciones de memoria contiguas. En cambio, si el recorrido es por columnas, habrá que ir calculando y realizando saltos para trabajar con los elementos de la matriz.

Además, como era de esperar, las operaciones de lectura son mucho menos costosas que las operaciones de escritura, tal y como pasa con los discos duros.